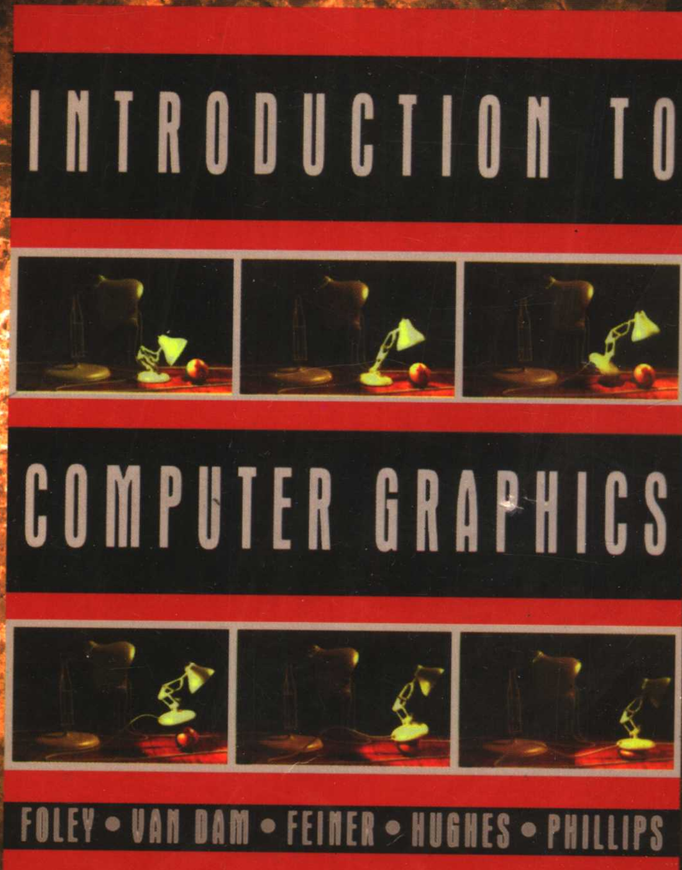
PEARSON
Addison Wesley

计算机科学丛书

计算机图形学导论

(美) James D. Foley Andries van Dam Steven K. Feiner John F. Hughes Richard L. Phillips 著
董士海 唐泽圣 李华 吴恩华 汪国平 等译



Introduction to Computer Graphics

机械工业出版社
China Machine Press

VISIT…

LANZAROTE Caliente.com

图形学领域的权威之作《计算机图形学原理及实践——C语言描述》是经典的计算机图形学教材，本书是《计算机图形学原理及实践——C语言描述》一书的简写版。本书主要面向计算机图形学的初学者和没有相关技术背景的读者，主题包括基本图形学编程、硬件及应用，二维和三维图形学的重要算法，同时还包括大量的习题、示例和彩色插图。本书的程序代码全部采用C语言编写，增强了可用性。

本书非常适合作为计算机专业计算机图形学课程的教材。

作者简介

James D. Foley 于密歇根大学获得博士学位，是佐治亚理工学院教授，图形学、可视化及可用性研究中心创始人，现任该中心主任。他还是ACM、ACM SIGGRAPH、ACM SIGCHI、IEEE成员。

Andries van Dam 于宾夕法尼亚大学获得博士学位，是布朗大学计算机科学系创始人之一，而且是该系的首任系主任，现为该系教授。他是IEEE计算机学会及ACM成员，ACM SIGGRAPH的创始人之一。

Steven K. Feiner 于布朗大学获得博士学位，是哥伦比亚大学计算机科学系副教授，负责计算机图形学组。他也是ACM SIGGRAPH和IEEE计算机学会成员。

John F. Hughes 于加利福尼亚大学伯克利分校获得博士学位，是布朗大学计算机科学和数学系教授，他与Andries van Dam共同负责计算机图形学组。他也是ACM SIGGRAPH和IEEE 计算机学会成员。

Richard L. Phillips 于密歇根大学获得博士学位，是密歇根大学电子与计算机工程系退休教授。他现在是Los Alamos国家实验室科学家，也是ACM和IEEE成员。

ISBN 7-111-14147-4

9 787111 141471

封面设计 陈子平

HZ BOOKS
华章图书

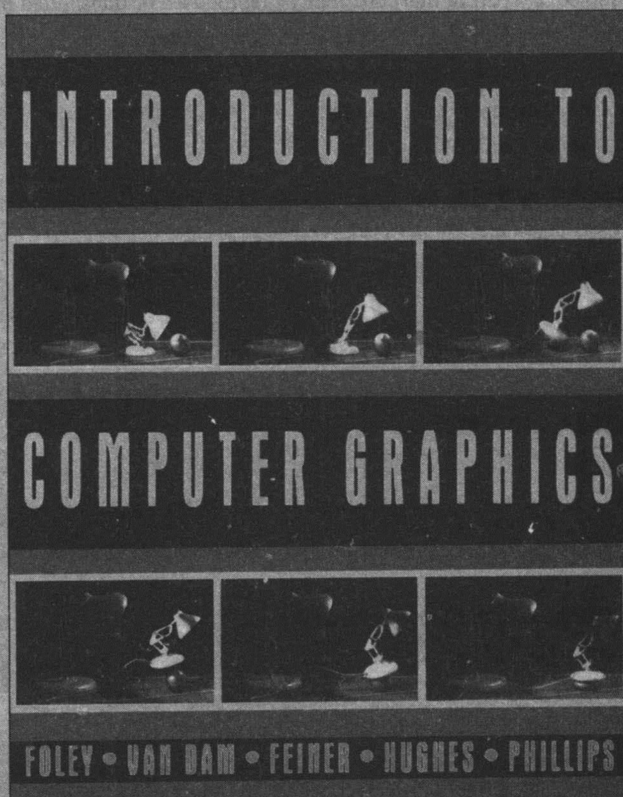
网上购书：www.china-pub.com
北京市西城区百万庄南街1号 100037
读者服务热线：(010)68995259，68995264
读者服务信箱：hzedu@hzbook.com
http://www.hzbook.com

ISBN 7-111-14147-4/TP · 3506
定价：45.00 元

计 算 机 科 学 丛 书

计算机图形学导论

(美) James D. Foley Andries van Dam Steven K. Feiner John F. Hughes Richard L. Phillips 著
董士海 唐泽圣 李华 吴恩华 汪国平 等译



Introduction to Computer Graphics

机械工业出版社
China Machine Press

本书是通过对经典的计算机图形学教材《计算机图形学原理及实践——C语言描述（第2版）》的部分高级主题进行删节和修改而形成的简写版。本书包括SRGP的编程、二维图元的基本光栅图形学算法、图形硬件、几何变换、三维空间的观察、对象的层次结构和SPHIGS系统、输入设备、交互技术与交互任务、曲线与曲面的表示、实体造型、消色光与彩色光、可视图像的真实性、可见面判定以及光照与明暗处理等内容。书中不但介绍了基本图形学编程、硬件及应用，二维和三维图形学的重要算法，同时还包括大量的习题、示例和彩色插图。本书的程序代码全部采用C语言编写。

本书非常适合作为计算机专业计算机图形学课程的教材。

Authorized translation from the English language edition entitled *Introduction to Computer Graphics* (ISBN: 0-201-60921-5) by James D. Foley et al., published by Pearson Education, Inc, publishing as Addison-Wesley , Copyright © 1994, 1990 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanic, including photocopying, recording, or by any information storage retrieval system, without permission of Pearson Education, Inc.

Chinese simplified language edition published by China Machine Press.

Copyright © 2004 by China Machine Press.

本书中文简体字版由美国Pearson Education培生教育出版集团授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2003-8109

图书在版编目（CIP）数据

计算机图形学导论 /（美）福利（Foley, J. D.）等著；董士海等译. - 北京：机械工业出版社，2004.5

（计算机科学丛书）

书名原文：Introduction to Computer Graphics

ISBN 7-111-14147-4

Ⅰ.计… Ⅱ.①福… ②董… Ⅲ.①计算机图形学-高等学校-教材 Ⅳ. TP391.41

中国版本图书馆CIP数据核字（2004）第019406号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：杨海玲

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2004 年5月第1版第1次印刷

787mm × 1092mm 1/16 · 27.25印张（彩插1.25印张）

印数：0 001-5 000册

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

译 序

本书是Foley、van Dam、Feiner 及 Hughes合著的《计算机图形学原理及实践——C语言描述（第2版）》的改编本。本书既保持了《计算机图形学原理及实践》这本图形学经典著作的最重要特色：对图形学理论（包括数学基础、人机交互、几何建模、真实感绘制等）的深入阐述，所涵盖的图形学内容尽可能先进，大量的习题、例子、图片和参考文献；又对原书的大量高级内容（主要是原书第17章以后的高级硬件结构、高级几何及光栅算法、高级建模技术等）进行了缩减和改写，以便适应大批基本读者的学习需求。应该指出的是本书还花了适当的篇幅增加了若干详尽而有特色的例子（如例2.1，例3.1，例5.1，例6.1，例6.2，例7.1，例9.1，例9.2等）。现在，贡献给读者的这本《计算机图形学导论》更加适合作为不同类型“计算机图形学”课程的教材和关注图形技术发展的广大读者的自学读物。

本书的中文版是在我国计算机图形学专家董士海、唐泽圣、李华、吴恩华、汪国平等合译的《计算机图形学原理及实践——C语言描述（第2版）》（2004年机械工业出版社出版）一书译稿基础上，由本人对照英文版原文进行补译、改正及整理而成。《计算机图形学原理及实践——C语言描述（第2版）》的译者为本书的翻译做了最重要的基础工作。衷心感谢机械工业出版社华章公司编辑所做的大量细致的补译、选辑及编辑加工工作。译文中欠妥和纰漏之处恐难避免，恳请读者不吝赐教和指正。

董士海

于北京大学

2004年4月8日

译者简介

董士海 北京大学信息科学技术学院教授，博士生导师。中国计算机学会虚拟现实和可视化专业委员会副主任，中国图像图形学会理事，中国人工智能学会理事，若干学报编委。1982年美国马里兰大学访问学者。曾参加我国首台百万次计算机、汉字激光照排系统、C软件环境、青鸟软件环境等研制。主持完成三维图形包、超媒体、多通道用户界面等国家攻关及自然基金重点项目。获政府特殊津贴、北京大学、国家教委、国家科技进步奖等奖励。已出版《计算机软件工程环境和软件工具》、《计算机用户界面及工具》、《人机交互和多通道用户界面》等专著，发表论文一百余篇。研究兴趣为图形学、人机交互及虚拟现实等。

唐泽圣 清华大学计算机科学与技术系教授，博士生导师。现任澳门科技大学副校长，信息科技学院院长，教授。美国IEEE 学会高级会员。澳门电脑学会名誉会长。曾任中国计算机学会理事长。长期从事计算机图形学及科学计算可视化方面的研究工作。主持、参加并完成了中国国家自然科学基金重点项目“科学计算可视化的理论和方法研究”及“多维动态地理信息系统关键技术研究”等课题。完成了国家自然科学基金一般项目“三维数据场整体显示技术的研究”及国家863高技术研究项目“计算机辅助立体定向神经外科手术系统”等课题。曾获国家科技进步三等奖一项、省、部级科技进步奖多项。有著作四部，发表论文100余篇。目前的研究方向有：计算机图形学、科学计算可视化，虚拟现实及地理信息系统等。

李华 1982年毕业于北京航空学院，1986年在该校获硕士学位，1989年在中科院计算所获博士学位。曾在航空部601所从事飞机设计，先后在德国、日本留学。现为中国科学院计算技术研究所研究员，博士生导师，中国计算机学会理事，中国计算机学会计算机辅助设计与图形学专业委员会主任，中国工程图学会常务理事，中国图像图形学会常务理事，美国IEEE北京分会执行委员会委员、技术委员会主席，《计算机辅助设计与图形学学报》副主编，《软件学报》、《中国图像图形学报》和《工程图学学报》编委。主要研究领域为计算机图形学和计算机辅助几何设计。专业研究兴趣包括虚拟现实、科学计算可视化、医学图像处理、数字化虚拟人、人体运动仿真等。获政府特殊津贴、国家科技进步奖等奖励。

吴恩华 男，1970年于北京清华大学计算数学专业本科毕业，其后从1970年至1980年在清华大学计算机科学与工程系进行教学和科研工作，1984年于英国曼彻斯特大学计算机科学系获博士学位。从1985年起在中国科学院软件研究所从事科研工作，历任研究室主任、基础和高技术研究部主任兼科研所长助理、研究所学位委员会主席等。于1997年9月起兼任澳门大学科技学院教授，并自2002年3月起担任澳大电脑与资讯科学系主任。于1980年起从事计算机图形学研究工作，并于1993年起聘为博士生导师，在国内外发表论文100余篇。主要研究领域是真实感图形、虚拟现实、科学计算可视化等。于2002年获CHINAGRAPH中国计算机图形学杰出奖。现受邀分别担任《计算机辅助设计与图形学学报》与《Journal of Computer Science and Technology》主编与副主编，并从2001年起受邀担任《The Journal of Visualization and Computer Animation》和《International Journal of Image and Graphics》编委。IEEE和 ACM会员。

汪国平 博士，教授，博士生导师，现任北京大学信息科学技术学院软件研究所副所长，人机交互与多媒体实验室主任，兼任中国图像图形学会常务理事，中国图像图形学会多媒体专业委员会和虚拟现实专业委员会委员，中国计算机学会虚拟现实和可视化专业委员会常务委员以及CADCG专业委员会委员，《计算机辅助设计与图形学学报》编委等。主要从事计算机图形学、人机交互与虚拟现实、网络多媒体等领域的教学与研究。近几年来负责国家级和省部级等各种项目20余项，负责分布式复杂虚拟场景构造和勘查平台，多媒体内容交互制作工具和同步合成系统，视频会议系统和视频点播系统等多个支撑系统的研制。申请国家发明专利4项，在国内外发表学术论文50余篇。

前 言

本书是Foley、van Dam、Feiner 及 Hughes合著《计算机图形学原理及实践——C语言描述（第2版）》（以下简称《计算机图形学》）的改编本。本书通过删节、修改《计算机图形学》一书中那些内容广泛的教学和参考部分编写而成的，以便适应于不同课程和不同专业的需要。本书大概是《计算机图形学》的一半，但它不仅仅是《计算机图形学》的缩略版本。事实上，本书是重新改写的，并在某些情况下注意用不同的方法进行说明，以便适应读者的需要。

本书可用于任何四年制大学和学院的计算机图形学课程（一到两学期）。在具备少量数学知识的前提下，也可用于两年制学校、成人大学的一学期课程。对于想要学习发展迅速和令人激动的这一学科基础知识的专业人员来说，不论想成为一名计算机图形技术方面的从业者，还是只想增加对计算机图形技术广泛应用的了解，本书都是一本理想的书籍。

从更新或更全面的角度来看，本书并不希望取代《计算机图形学》一书。但是，《计算机图形学》一书的某些章节的内容由于此领域正以飞快的速度发展、因材料较旧而不再使用或者因其硬件性能和成本而被更新了。在《计算机图形学》（1990年出版）的第4章里可以找到这样一个例子："图形工作站通常包括一个至少执行几百万条指令/秒（MIPS）的CPU"，而现在已修改为20~100 MIPS，以反映当前的事实。

本书的其他主要差别和强项是：

- 在完整的工作程序和用伪代码书写的程序段中，全书均使用了近代计算机语言ANSI C。使用C是符合当今教学和专业实践的，尤其是在图形学领域。
- 作为本书使用C的一个直接好处，SRGP及SPHIGS软件包与书中使用的代码功能和数据类型两者之间，现在是一一对应的。
- 上面提到的SPHIGS软件包，已经充分地增强了许多新的特性，如多光源、改进后的绘制以及用于更好交互操作的改进后的拾取校正。
- 本书给出了几个有特色的例子，其中有些是十分详尽的。这些例子总体上是放在用来最好地展示很难的概念的那些章节。例如，用于交互地定义Bézier三次参数曲线的完整工作程序。
- 本书提供大量例子、图表及相关参考文献来介绍计算机图形学在新兴多媒体领域中的重要性。
- 在本书的第5章中增加了数学基础知识一节。这一节为读者提供了足够的信息，以便读者理解和使用本书中与数学相关的所有材料。

可选的教学大纲

可以有多种路径来阅读全书。这里仅提供其中的一些，但读者完全可以灵活地选择以适合自己的情况。为了学习，甚至可改变次序。例如，第4章提供的有关硬件的材料，在教学中完全可以提前或者延后。

着重于2D图形学的最少一学期的课程安排　这门课程适合于两年制或四年制大学的学生，主要目标是提供2D图形学各种元素的总论。

章	节
1	全部
2	2.1节~2.2节
3	3.1节~3.3节，3.9节~3.9.3节
4	4.1节，4.2节，4.3节及4.5节
5	5.1节（如合适），5.2节，5.3节，5.4节
6	6.1节，6.2节，6.3节，6.4.1节，6.4.2节
8	全部
9	9.1节，9.2.1节~9.2.3节
11	11.1节~11.2节
12	选择阅读与演示高级能力相关的部分

提供2D及3D图形学概论的一学期课程安排　这个大纲适合于有较好数学基础知识的读者，为学习图形学提供了坚实的基础。

章	节
1	全部
2	全部
3	3.1节~3.5节，3.8节~3.11节，3.14节~3.15节
4	4.1节，4.2节，4.3节及4.5节
5	5.1节（如合适），5.2节~5.5节，5.7节，5.8节
6	6.1节~6.5节，6.6节（除6.6.4节外），6.7节
7	7.1节~7.5节，7.10节及7.11.6节
8	全部
9	9.1节，9.2.1节~9.2.3节，9.2.7节，9.3.1节~9.3.2节
11	全部
12	全部
13	13.1节~13.2节，可能13.4节
14	14.1节~14.2节，可能14.5节~14.7节

覆盖2D及3D图形学、建模、绘制的两学期课程安排　本书的所有各章（可从第9章和第10章中略去若干主题），加上从《计算机图形学》一书中选取的主题。

由于本书的许多读者会对《计算机图形学》一书的高级和全面内容感兴趣，所以《计算机图形学原理及实践——C语言描述（第2版）》一书的前言也列在后面。读者可找到该书重要特性的讨论和基于该书来架构课程的建议。

致谢

首先，我要说明《计算机图形学》一书的全部作者均在某种程度上参与了本书的改编。我将承担改编过程中任何新错误的全部责任。

David Sklar是《计算机图形学》一书的客座作者，他贡献了那本书里的许多材料，保存在本书的第2章和第7章中。他也帮助我找到了那本书里的计算机代码和艺术图片的电子版。

编辑Peter Gordon在这个项目的全过程经常及时、审慎、沉着地给我以指点。生产监督Jim Rigney花费了许多时间教我“职业的诀窍”。

还有许多人在本书的不同方面给予了帮助。他们是Yvonne Martinez、Chuck Hansen、Tom Rokicki、David Cortesi、Janick (J.) Bergeron、Henry McGilton、Greg Cockroft、Mike Hawley、Ed Catmull、Loren Carpenter、Harold Borkin、Alan Paeth、Jim White 及Bert Herzog。

特别要感谢美国新墨西哥大学的Ed Angel和他出色的学生们，他们在1992年秋为本书的第一个草稿进行了β测试。

最后，如果没有D. C. 本书将永远不能问世。

R. L. P.

于Santa Fe, N. M.

作者简介

James D. Foley（密歇根大学博士）是美国佐治亚理工学院计算机科学系和电子工程系教授，图形学、可视化及可用性研究中心主任、创始人。他和Andries van Dam是《Fundamentals of Interactive Computer Graphics》一书的作者。他是ACM、ACM SIGGRAPH、ACM SIGCHI、the Human Factors Society、IEEE、IEEE 计算机学会会员、《ACM Transactions on Graphics》主编。《Computers and Graphics》和许多著名杂志的编委。研究领域是用户界面设计环境（UIDE，一种基于模型的用户界面开发工具）、用户界面软件、信息可视化、多媒体和用户界面中的人的因素。他还是IEEE会员，Phi Beta Kappa、Tau Beta Pi和Sigma Xi的成员。

Andries van Dam（宾夕法尼亚大学博士）是美国布朗大学计算机科学系创始人和首任系主任，目前是L. Herbert Ballou大学和布朗大学计算机科学系教授，BLOC Development 和Electronic Book Technologies公司的高级顾问科学家，ShoGraphics和Microsoft公司的技术顾问委员会成员，IEEE 计算机学会会员和ACM会员，也是 ACM SIGGRAPH创始人之一。van Dam帮助创建了《Computer Graphics and Image Processing》和《ACM Transactions on Graphics》杂志，并曾任其编辑。他和James Foley一起是《Fundamentals of Interactive Computer Graphics》一书的作者，和David Niguidula一起是《Pascal on the Macintosh: A Graphical Approach》一书的作者。已发表了80余篇论文。1984年获IEEE Centennial Medal奖，1988年获罗得岛州政府的科学技术奖，1990年获NCGA学术奖；1991年获SIGGRAPH Steven A. Coons奖。他的研究领域包括超媒体、电子书和用于教学研究的高性能工作站。

Steven K. Feiner（布朗大学博士）是美国哥伦比亚大学计算机科学系副教授，该校的计算机图形学和用户界面实验室的负责人。当前研究领域是图片合成、人工智能应用于计算机图形学、用户界面、动画、虚拟世界及超媒体系统。他也潜心于图形用户界面自动化设计和布局的基于知识系统的开发。《Electronic Publishing》和《ACM Transactions on Information Systems》杂志的编委，ACM SIGGRAPH 和 IEEE 计算机学会的会员。1991年获ONR Young Investigator奖。发表论文40余篇，并在许多讲座和讨论会上作报告。

John F. Hughes（加利福尼亚大学伯克利分校博士）是美国布朗大学计算机科学系副教授，与Andries van Dam共同领导该校的计算机图形学研究组。研究领域是：为科学和数学可视化进行的将数学应用于计算机图形学、自动计算机动画、计算机图形学基础和交互图示。他是ACM SIGGRAPH 和 IEEE计算机学会的会员。最近的论文发表在《Computer Graphics》和《Topology》上。有关球体外翻的著作已在《Science News》的封面文章中描述。

Richard L. Phillips（密歇根大学博士）是本书的主要负责人。美国密歇根大学电机和计算机工程系、宇航工程系的退休教授。作为计算机辅助工程网络的创立者和主任，他帮助创建了该校工程学院的学生和教职员建立的几百个节点的工作站网络，他还是信息技术集成中心的创立者和主任。当前，他是Los Alamos国家实验室的技术人员，在那里，他的研究领域是科学可视化、多媒体工作站、分布式计算和多媒体数字出版。他是IEEE和ACM的会员以及《Computers and Graphics》的编委。

《计算机图形学原理及实践——C语言描述（第2版）》前言

交互式图形学的时代已经到来。就在不久以前这还是一项需要昂贵显示器硬件、大量计算机资源和独特软件的深奥的专业领域。然而在过去的几年中，随着硬件性能价格比的大幅提升（如配有标准图形终端的个人计算机）以及高端的与设备无关的图形程序包的开发，图形编程已经变得简捷、合理。交互式图形学如今已经可以为我们提供图形通信手段，并成为人机交互的主要工具。（节选自《交互式计算机图形学基础》的前言。*Fundamentals of Interactive Computer Graphics*, James Foley, Andries van Dam, 1982。）

这一断言在Apple公司的Macintosh机、IBM的PC机及其他类似产品引发计算机文化革命之前就做出了。现在，就算是没上学的孩子都对交互图形技术非常熟悉，比如窗口控制、用鼠标选取菜单和图标。图形用户界面可以使新手迅速变得老练，没有图形界面的计算机已经越来越罕见了。

随着交互式图形学在用户界面和数据可视化方面的广泛应用，三维物体的绘制技术也变得越来越真实，运用这些技术生成的广告片和电影特技无所不在。20世纪80年代初期尚处于实验阶段的技术目前已变得很普通，而更加令人叹为观止的“照片真实感”技术也马上就要到来。曾经需要花上几个小时才能生成一帧的伪真实感动画，如今在个人计算机上可以以10帧/秒以上的速度生成。1981年的“实时”向量显示器可以显示没有经过隐藏线消除的几万个向量组成的移动线框物体；而1990年的实时光栅显示器不仅可以显示同样的线框物体，而且可以显示由十万多个三角形面片组成的、采用Gourand或者Phong明暗处理以及完全的隐藏面消除的移动物体。这些高性能系统能提供实时的纹理映射、反走样、云雾和烟尘的大气衰减以及其他特殊效果。

图形软件的标准同样较第1版时明显进步了很多。第1版的SGP软件包基于的SIGGRAPH Core’79软件包、直视存储管和刷新向量显示器现在基本上都已经消失了。支持结构层次存储和编辑的更加强大的PHIGS软件包已经成为ANSI和ISO的标准，广泛应用于科学和工程的实时几何图形制作中，与其相伴的PHIGS + 支持光照、明暗处理、曲线和曲面。官方的图形标准中补充了许多事实标准，如Apple公司的QuickDraw，X Window的Xlib二维整型光栅图形包和Silicon Graphics公司的GL三维图形库。Pixar公司的照片真实感的RenderMan软件和硬拷贝页面和屏幕图像描述的PostScript图形解释器同样也得到了广泛的应用。更好的图形软件的使用使得对用户界面的“视感”得到巨大改善，我们可以期待增加对三维效果的利用，使我们对信息的管理、表现、检索和漫游提供新的想像空间和形象，必是出于审美的考虑。

也许图形学中最重要的发展方向是对物体建模技术的愈加重视，不仅仅是生成这些物体的画面，同时人们对如何描述随时间变化的三维几何物体的几何特性和行为产生了更大的兴趣。这样，图形学在建模和绘制中就更加关注模拟、动画及“回归物理过程”，试图使创作出的物体尽可能真实。

当图形工具变得越来越复杂时，我们就需要能够有效地运用它们。绘制不再是瓶颈，所以研究人员开始尝试用人工智能技术帮助进行物体建模设计、运动规划、二维及三维物体的有效图形表示的布局。

当今图形学技术发展迅猛，任何一本参考书都必须不断地进行更新和扩充才能跟上这样的发展。本书基本上对《交互式计算机图形学基础》做了总体重写，尽管页数几乎翻番，我们仍然不得不省略大量内容。

本书和第1版的主要区别如下：

- 向量图形学的定位由光栅图形学所代替。
- 原来简单的二维浮点图形包（SGP）被SRGP和SPHIGS代替，体现了交互图形程序设计的两大主要流派。SRGP结合了QuickDraw和Xlib二维整型光栅图形包的特征；基于PHIGS的SPHIGS提供了具有层次显示列表的三维浮点包的基本特性。本书描述了如何应用这些标准图形包进行编程，并且同时讲述了这些图形包对于基本裁剪算法、扫描转换算法、观察算法和显示列表遍历算法等基本功能的实现细节。
- 在更深的层次上讨论了用户界面问题，包括二维桌面隐喻和三维交互设备。
- 对建模的讨论包括了NURB（非均匀有理B样条）曲线和曲面，实体造型以及高级建模技术，如基于物理的建模、过程化模型、分形、L文法系统和粒子系统等。
- 对绘制技术的讨论增加了对反走样的详细讨论，以及可见面判定、光照和明暗处理，包括基于物理的光照模型、光线跟踪、辐射度等。
- 增加了高级光栅图形体系结构和算法的内容，包括裁剪、复杂图元的扫描转换、简单的图像处理操作等。
- 增加了对动画的简单讨论。

阅读本书无须具备图形学的基础知识，只需要了解一定的C语言编程技术、基本的数据结构和算法、计算机体系结构和简单的线性代数。附录中列出了阅读本书必需的数学基础。本书的全部内容可在两个学期内讲授，但是书中内容划分为几个部分，也可以有选择地进行讲授。因此，读者可以根据自己的需要选择学习，由浅入深。本书可以大致分为下面几个部分：

基本知识

第1章对历史进行了简要回顾，并对硬件、软件、应用程序的最基本问题做了讨论。第2和3章讲述了简单的二维整型图形包SRGP的应用和实现方法。第4章介绍了图形硬件，包括如何应用硬件完成前面几章中提到的操作。第5和6章通过矩阵的方法介绍了在平面和三维空间中进行变换的思想，如何应用齐次坐标来统一线性变换和仿射变换，三维视图的描述，包括从任意视见体到标准视见体的变换。最后，第7章介绍了三维浮点层次图形包SPHIGS，通过一些基本的建模操作讲述了它的用法，SPHIGS是PHIGS标准的简化版本。这一章还讨论了PHIGS中可用的层次结构的优缺点以及使用这个图形软件包的应用程序的结构。

用户界面

第8~10章讲述了当前交互设备并讨论了用户界面设计的一些高层次问题，对当前流行的各种用户界面设计范型进行了介绍和比较。最后的用户界面软件一章讨论了窗口管理器、交互技术库和用户界面管理系统。

模型定义

第11和12章讲述了当前的几何建模技术：曲线和曲面的参数函数表示（尤其是三次样条函数）以及各种技术的实体表示，包括边界表示和CSG模型。第13章介绍了人类的颜色视觉系统以及各种颜色描述系统及它们之间的转换，同时也讨论了如何有效运用颜色的规则。

图像合成

在连续四章中的第一章，即第14章，讲述了从最早的向量绘图到最新的光照图形技术，人们

对真实感的探索过程。走样所引起的人为痕迹是光栅图形学中首要考虑的问题，这一章讨论了产生这些人为痕迹的原因和利用傅里叶变换的解决办法。第15章详细讨论了可见面判定的不同方法。第16章介绍了光照和明暗处理算法，本章的前半部分讨论了当前在流行的图形硬件中使用的算法而其余部分讨论了纹理、阴影、透明效果、反射、基于物理的光照模型、光线跟踪、辐射度，等等。第17章讲述了图像操纵（如像素图的缩放、错切、旋转）和图像存储技术（包括各种图像压缩技术）。

高级技术

最后四章对最新的图形学技术做了简介。第18章描述了当前的高端商用和研究用的图形硬件，本章由高性能图形体系结构的权威Steven Molnar和Henry Fuchs提供。第19章讲述了应用于如任意圆锥曲线的扫描转换、反走样文字生成、页描述语言实现（如PostScript）等任务的复杂光栅算法。最后两章对在高级建模和计算机动画领域中最重要的技术做了概述。

前两部分内容相对基础，可用于本科生的基础课程，随后的课程可使用其余各章的高级内容。当然也可以从各部分中抽取章节定制课程内容。

比如，以二维图形学为主的课程可以包括第1和2章，第3章中的简单扫描转换和裁剪，第4章中的概述、光栅体系结构和交互设备，第5章的齐次数学，6.1节~6.3节从“如何使用三维观察”的角度讲解了三维观察，由第8、9和10章组成的用户界面部分，以及由第14、15和16章组成的图像合成部分的引言和简单算法。

一门图形学概论课程可以包括第1、2章，第3章的简单算法，第4章中的光栅体系结构和交互设备，第5章，第6和7章的大部分内容以及SPHIGS。课程的后半部分包括第11和13章中的建模技术，第14、15和16章的图像合成，以及第20章中的部分高级内容。

以三维建模和绘制为重点的课程可以从第3章讲述扫描转换、线和多边形的裁剪、反走样的章节开始，然后进行到讲述变换和观察的数学基础的第5和6章，以及关于颜色的第13章，第14~16章的主要内容。也可以加入曲面和实体建模、第20章的高级建模技术、第21章的动画等高级内容。

图形包

由David Sklar设计的SRGP和SPHIGS图形包可以从出版商获取，可用于IBM PC (ISBN 0-201-54700-7)、Macintosh (ISBN 0-201-54701-5)和运行X11的UNIX 工作站，同时包括扫描转换、裁剪、观察等多种算法。

致谢

本书的完成离不开很多朋友和同事的辛勤工作。我们特别感谢那些为本书提供大量材料的人们，同样感谢对本书提出建议的很多同事。如有遗漏，敬请原谅。Katrina Avery和Lyn Dupré为本书的编辑做了大量工作，同时还有Debbie van Dam、Melissa Gold和Clare Campbell。我们特别感谢产品监督Bette Aaronson，艺术指导Joe Vetere和编辑Keith Wollman，他们不仅为本书做出了不懈努力，同时他们在过去五年所处的困境下所表现的耐心和幽默也为本书的完成做出了很大的贡献。

计算机图形学已经不是一个由四个主要作者和三位辅助作者可以完全掌握的领域，我们的同事、学生为本书提供了大量有价值的意见和建议，并发现了很多错误。下列人士对本书的一章或几章进行了仔细的技术性审读：John Airey, Kurt Akeley, Tom Banchoff, Brian Barsky, David Bates, Cliff Beshers, Gary Bishop, Peter Bono, Marvin Bunker, Bill Buxton, Edward Chang, Norman Chin, Michael F. Cohen, William Cowan, John Dennis, Tom Dewald, Scott Draves, Steve Drucker, Tom Duff, Richard Economy, David Ellsworth, Nick England, Jerry Farrell, Robin Forrest, Alain Fournier, Alan Freiden, Christina Gibbs, Melissa Gold, Mark Green, Cathleen

Greenberg, Margaret Hagen, Griff Hamlin, Pat Hanrahan, John Heidema, Rob Jacob, Abid Kamran, Mike Kappel, Henry Kaufman, Karen Kendler, David Kurlander, David Laidlaw, Keith Lantz, Hsien-Che Lee, Aaron Marcus, Nelson Max, Deborah Mayhew, Barbara Meier, Gary Meyer, Jim Michener Iakob Nielsen, Mark Nodine, Randy Pausch, Ari Requicha, David Rosenthal, David Salesin. Hanan Samet, James Sanford, James Sargent, Robin Schaufler, Robert Scheifler, John Schnizlein, MichaeI Shantzis, Ben Shneiderman, Ken Shoemake, Judith Schrier, John Sibert, Dave Simons, Jonathan Steinhart, Maureen Stone, Paul Strauss, Seth Tager, Peter Tanner, Brice Tebbs, Ben Trumbore, Yi Tso, Greg Turk, Jeff Vroom, Colin Ware, Gary Watkins, Chuck Weger, Kevin Weiler, Turner Whitted, George Wolbetg和Larry Wolff。

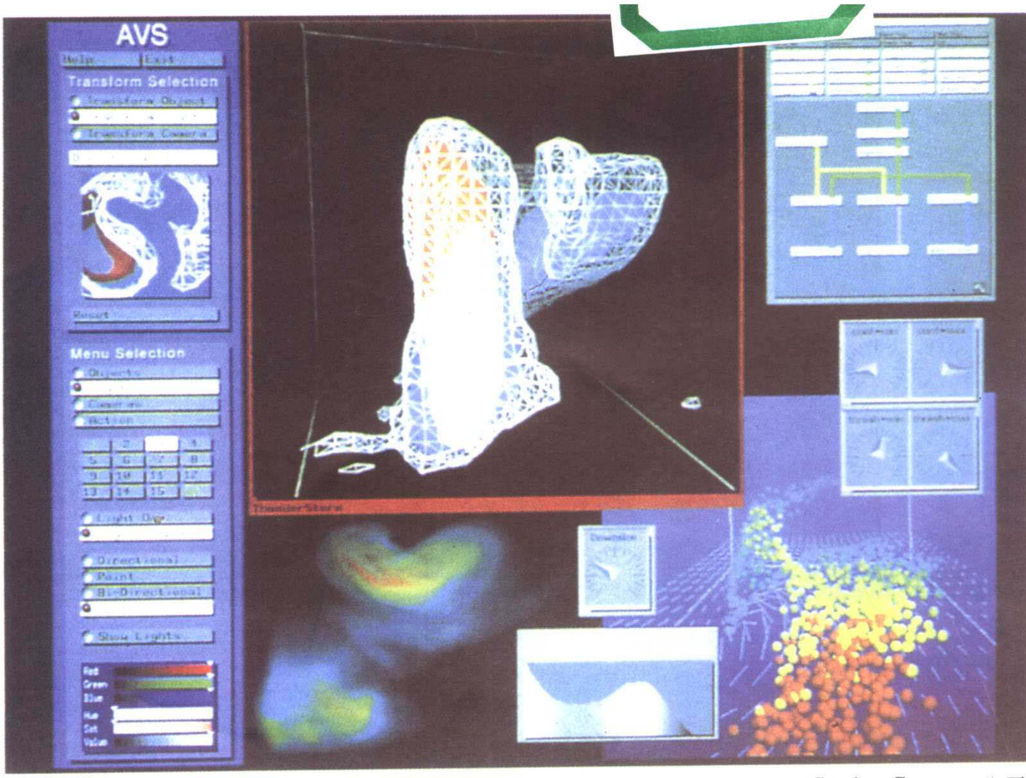
我们的一些同事，包括Jack Bresenham, Brian Barsky, Jerry Van Aken, Dilip Da Silva（建议对第3章采取统一的中点方法）和Don Hatfield，不仅对章节做了仔细阅读，同时对其中的算法提出了详细的建议。

Katrina Avery, Barbara Britten, Clare Campbell, Tina Cantor, Joyce Cavatoni, Louisa Hogan, Jenni Rodda和Debbie van Dam做了大量的文字处理工作。Dan Robbins, Scott Snibbe, Tina Cantor和Clare Campbell绘制了第1~3章的插图。其他插图由Beth Cobb, David Kurlander Allen Paeth和George Wolberg（在Peter Karp的帮助下）提供。彩图II-21~彩图II-37展示了绘制技术的进步，由Pixar的Thomas Williams和H.B. Siegel在M.W. Mantle的指导下应用Pixar的PhotoRealistic RenderMan软件完成。感谢Industrial Light & Magic提供激光扫描仪创建了彩图II-24~彩图II-37，感谢Norman Chin 为彩图II-30~彩图II-32计算了顶点法向量。L. Lu和Carles Castellsagué 为制作插图编写了程序。

Jeff Vogel实现了第3章的算法，他和Atul Butte检查了第2和7章的程序。在Ron Balsys, Scott Boyajian, Atul Butte, Alex Contovounesios和Scott Draves 的帮助下，David Sklar编写了Mac和X11上的SRGP 和SPHIGS。Randy Pausch和他的学生将这些包移植到了PC平台。

为了能够让读者获取多种算法的电子拷贝、提出习题、报告本书和SRGP/SPHIGS中的错误以及得到本书和软件的勘误表，我们已经安装了一个自动的电子邮件服务器。发一封主题为“Help”的电子邮件到graphtext@cs.brown.edu，就可收到当前可用的服务列表。

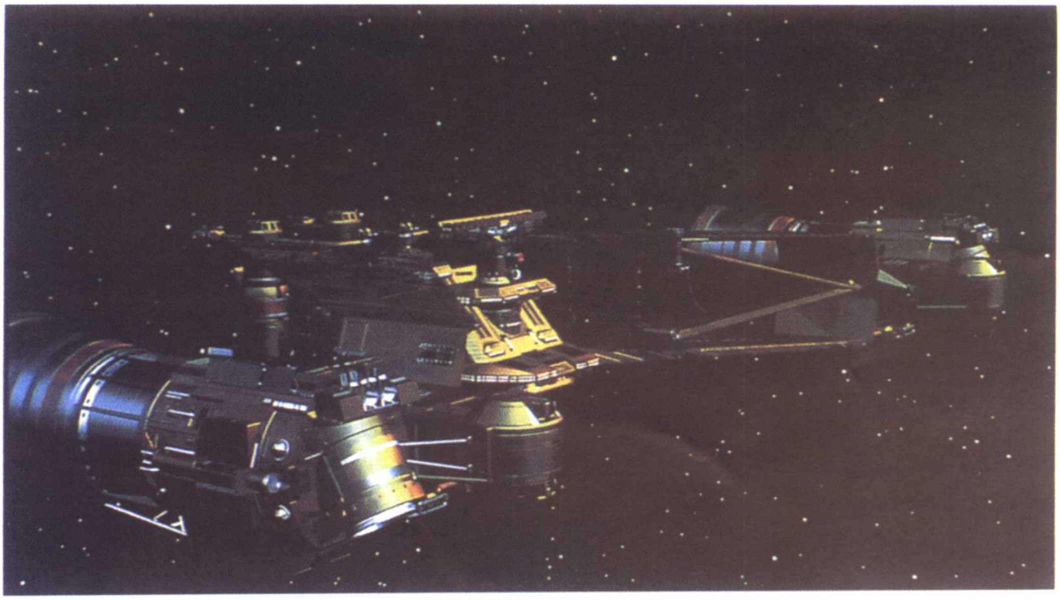
华盛顿州 J.D.F.
罗德岛州 A.v.D
纽约州 S.K.F.
罗德岛州 J.F.H.



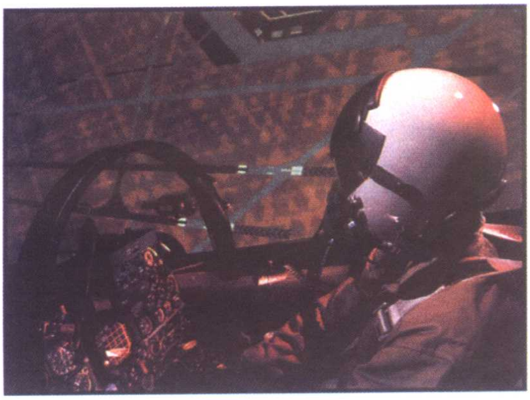
彩图1 强龙卷风，由Illinois大学NCSA的R.Wilhelmson、L.Wicker和C.Shaw提供。(Stardent Computer公司的应用可视化系统——AVS系统。)

彩图2 The Abyss——假足动物游戏（Twentieth Century Fox公司版权所有，1989，保留所有权利。经Industrial Light & Magic 的计算机图形部门许可。）

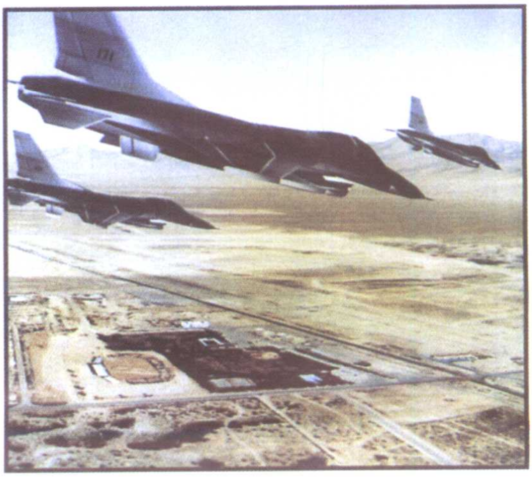




彩图3 《The Last Starfighter》中的指挥船。这艘经纹理映射后的船由超过450 000个多边形构成。(Digital Productions版权所有，1984。经G.Demos许可。)



a)



b)

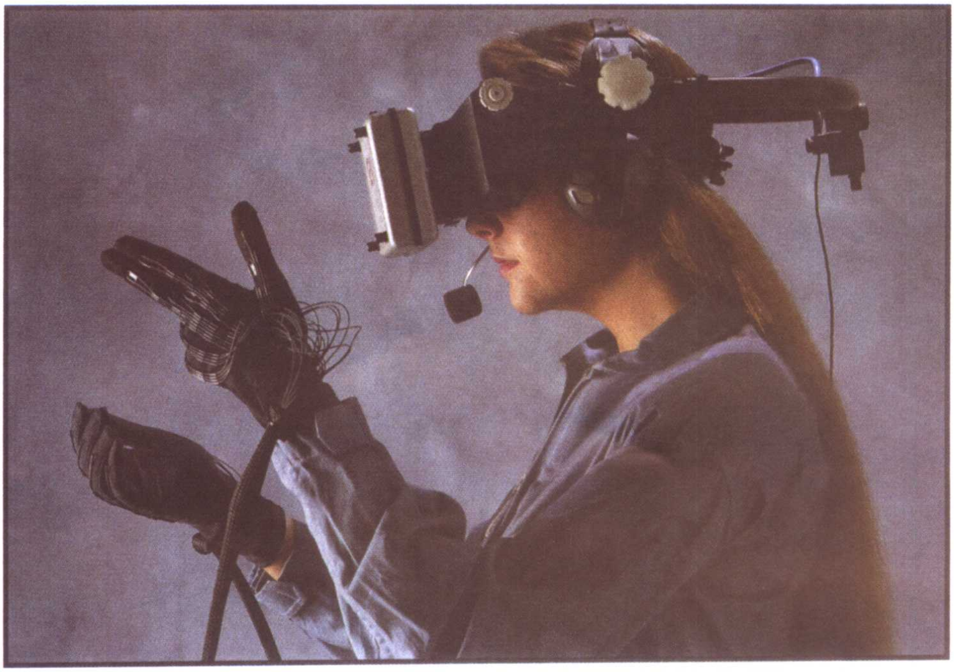
彩图4 a) 一架 F5飞行模拟器的驾驶舱；驾驶员视图投影在驾驶舱圆顶上。b) 从飞行模拟器驾驶舱里所见的视图。地形是用照片贴的纹理，而战斗机是采用几何建模的。(经通用电气公司的R.Economy许可使用。)



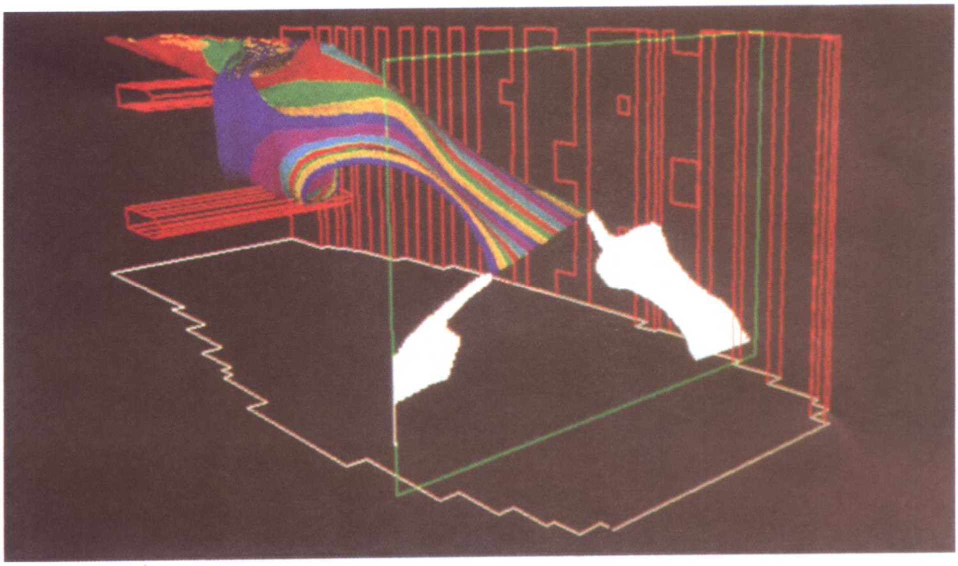
彩图5 Hard Drivin' 娱乐厅视频游戏。(经Atari Games公司提供，Atari Games公司版权所有，1988。)



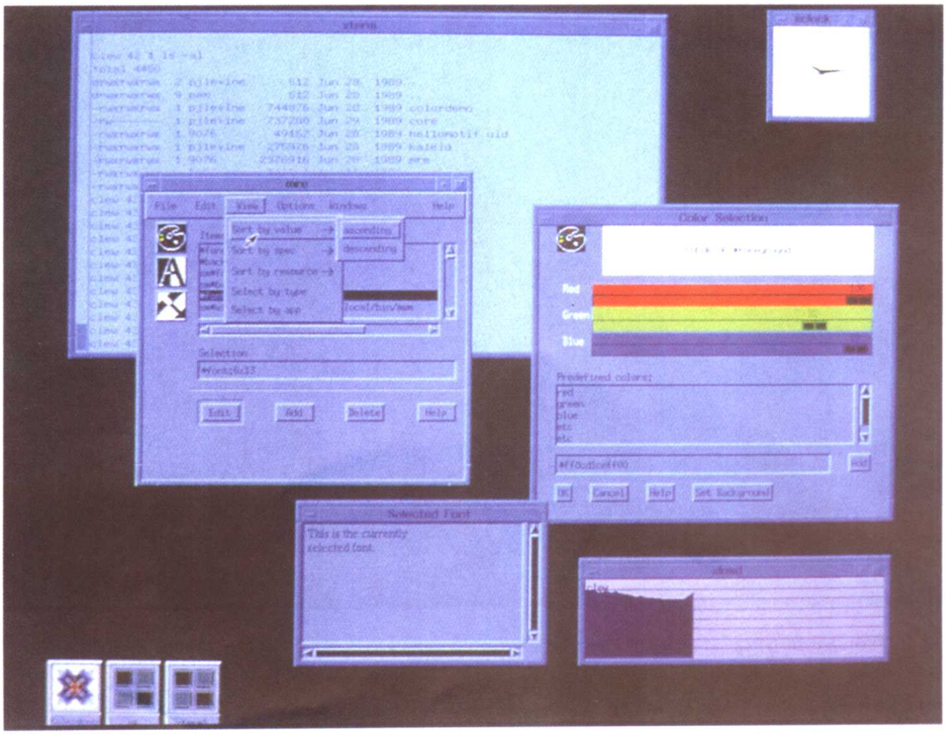
彩图 6 数据手套（DataGlove）（右）及其相应的计算机图像。数据手套用于测量手指位移量以及手的位置和朝向。相应的计算机图像也随之变化。(经VPL的Jaron Lanier许可。)



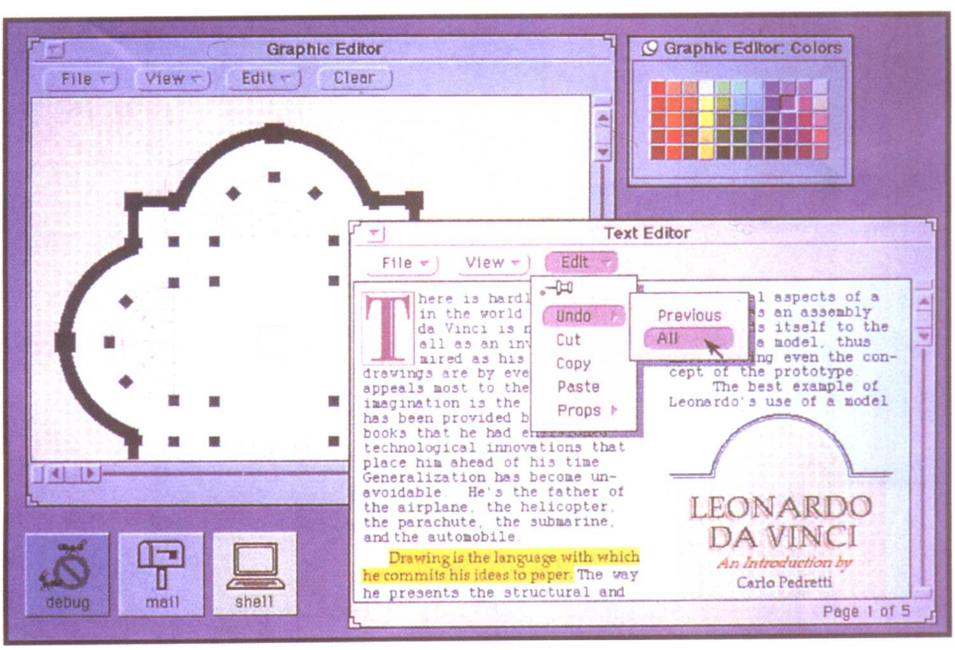
彩图 7 用户穿戴着一种头盔立体显示器、数据手套和用于发出命令的麦克风。这些设备用于构造虚拟现实环境，立体显示场景的变化通过头部的转动来实现，数据手套用于操纵计算机生成的物体。（经位于加利福尼亚州Moffett Field的NASA Ames研究中心的Michael McGreevey和Scott Fisher许可。）



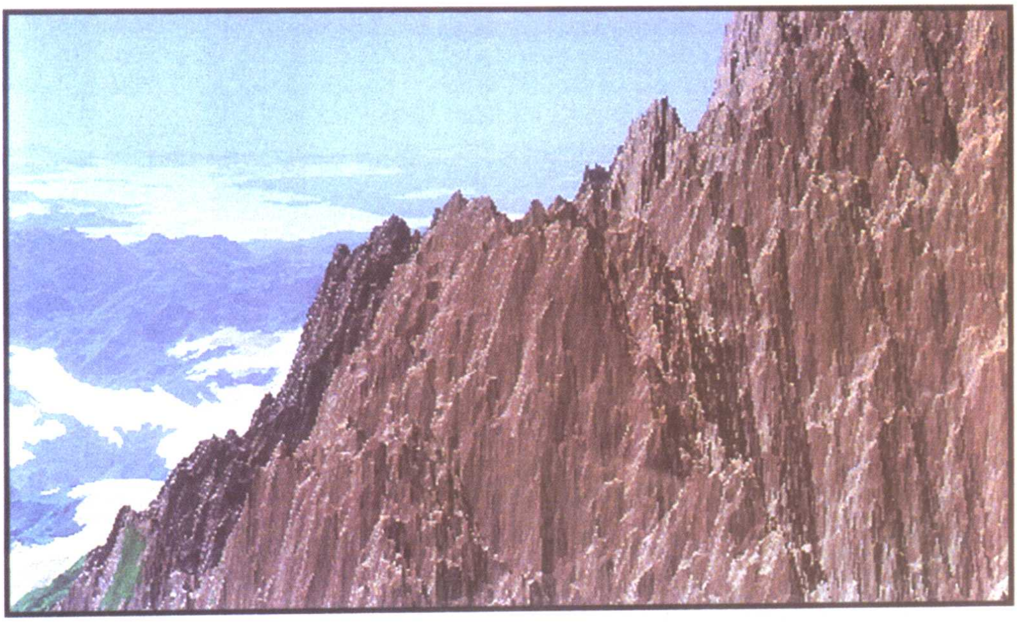
彩图 8 Krueger的Videotouch系统，其中用户用手的运动来操纵物体。手的轮廓与物体都显示在屏幕上，以提供自然的反馈。（经Artificial Reality公司许可。）



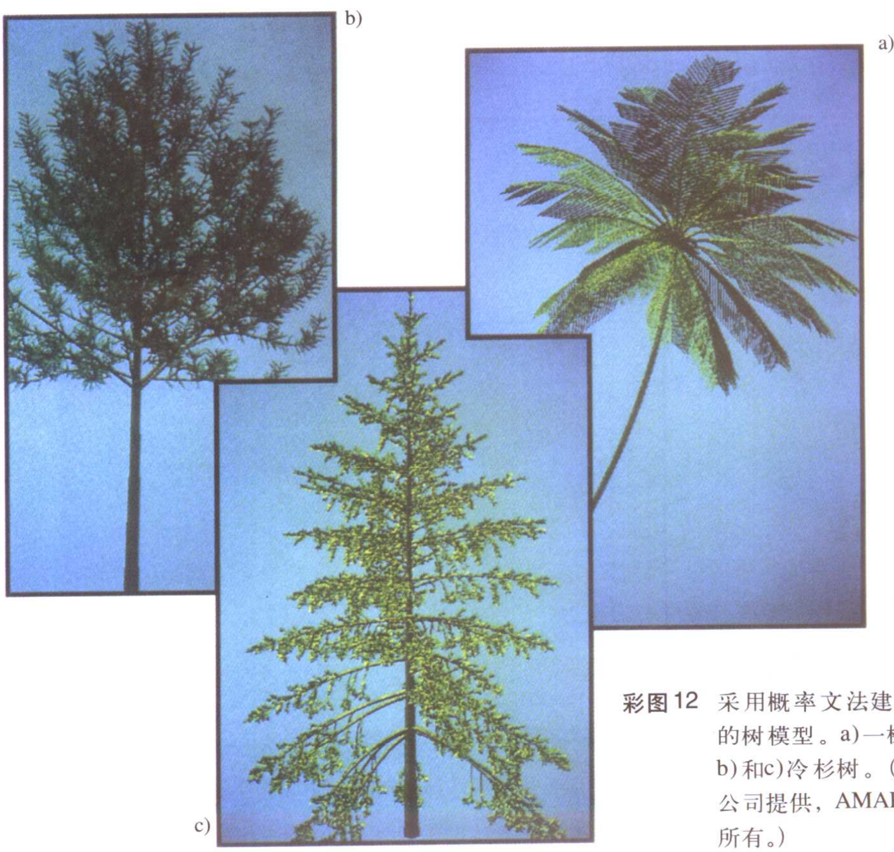
彩图9　OSF/Motif的用户界面，其中颜色滚动条用于定义不同窗口的颜色。注意，在按钮、菜单等边缘处使用明暗处理以创建三维效果。（经开放软件基金会（OSF）许可。）



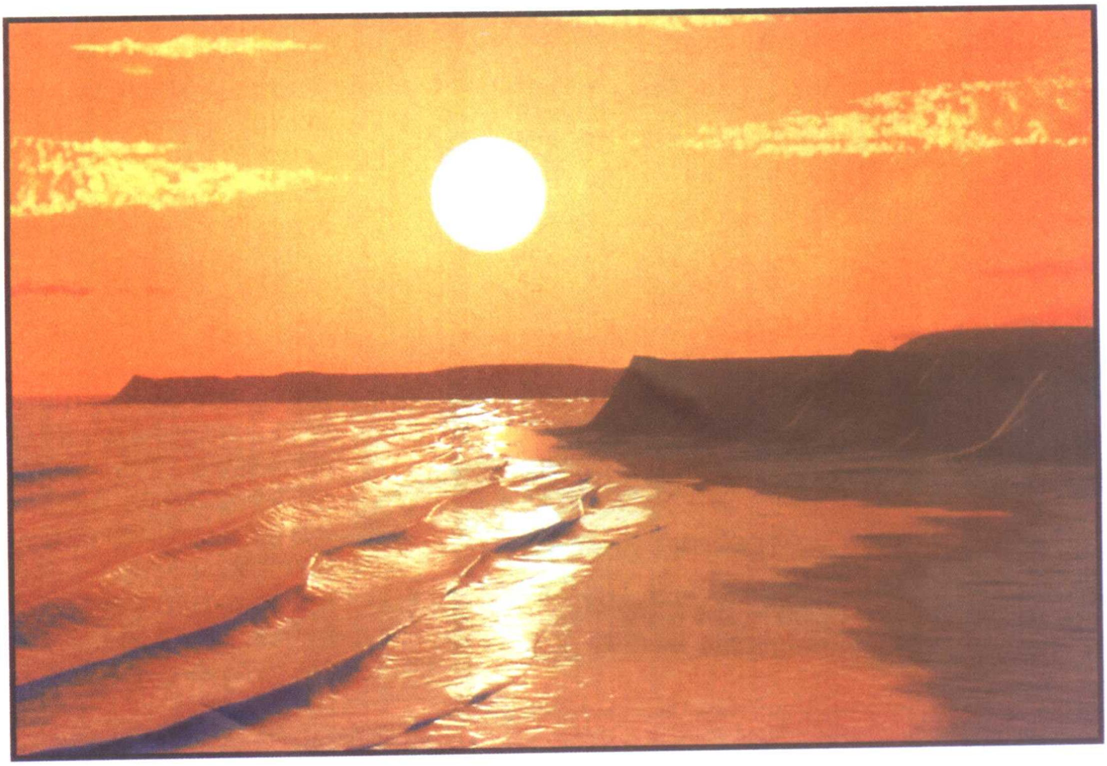
彩图10　OPEN LOOK的用户界面。黄色用于加亮所选择的文本，柔和的阴影用于窗口的背景和边缘。（经Sun Microsystems公司许可。）



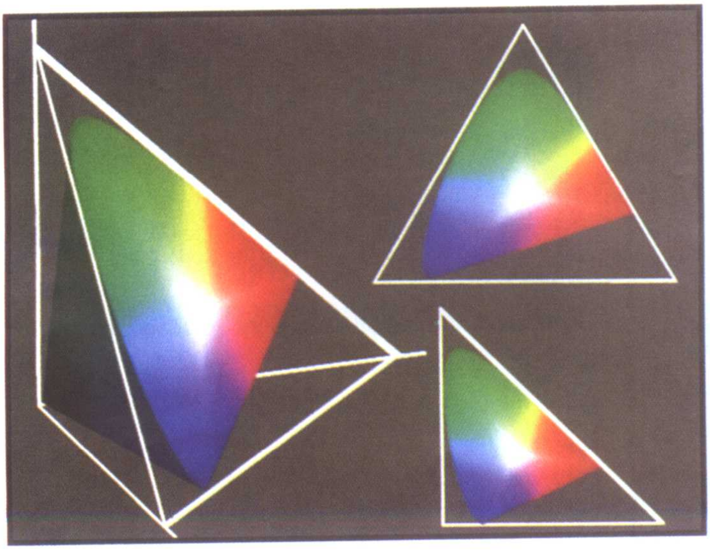
彩图 11　“Vol. Libre 山脊”：采用 Fournier-Fussell-Carpenter算法产生分形山。（经Loren Carpenter许可。）



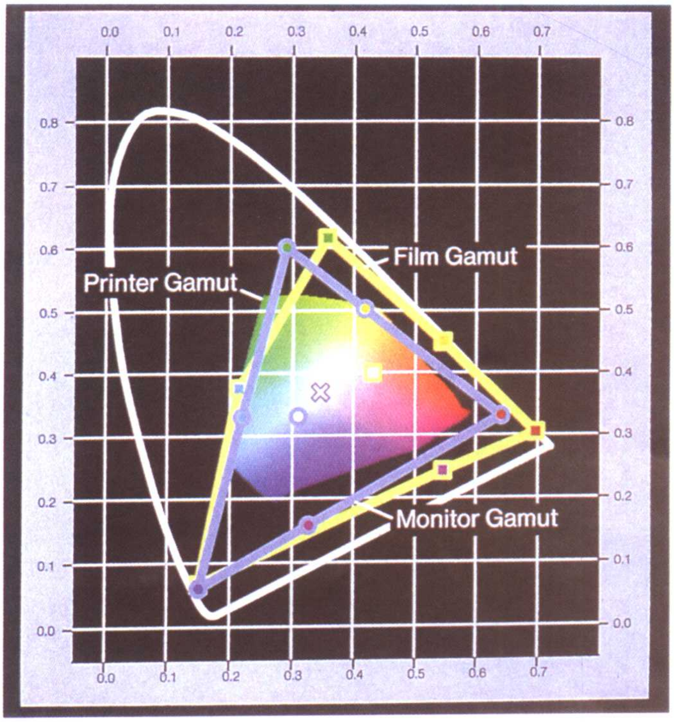
彩图 12　采用概率文法建模的简单的树模型。a)一棵棕榈树，b)和c)冷杉树。（由AMAP公司提供，AMAP公司版权所有。）



彩图13　日落的海滩。（经多伦多大学的Bill Reeves、Pixar和Alan Fournier许可。）

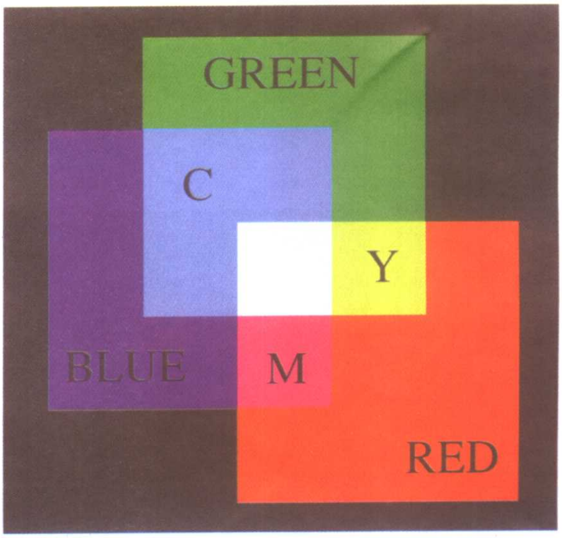


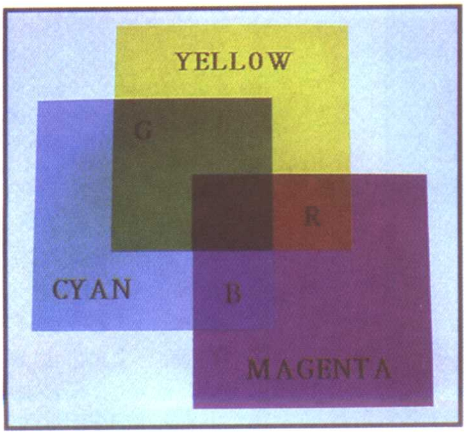
彩图14　CIE空间的$X+Y+Z=1$平面的几个视图。左边是嵌入在CIE空间内的平面，右上是该平面的垂直正面视图，右下是在（X,Y）平面（即$Z=0$平面）上的投影图，即是CIE色度图。（经布朗大学Barbara Meier许可。）



彩图15 显示了打印机、彩色监视器和电影胶片在CIE色度图上的典型颜色域。打印机的颜色代表一种在5000°K绝对温度的图形艺术光下测量的图形艺术技术基金会(GATF）的S.W.O.P. 标准颜色。彩色监视器的型号为Barco CTVM 3/51，它采用白色点集、工作在6500°K绝对温度下。电影胶片是 Kodak Ektachrome 5017 ISO 64，在CIE的A类条件（即近似在一种Tungsten灯光的2653°K绝对温度的黑体条件）下感光的颜色域。叉号、小圆及小方块分别表示打印机、彩色监视器及电影胶片的白点。（经Xerox PARC的M. Stone许可。电影胶片颜色域是由滑铁卢大学图形实验室的A. Paeth 测量的，可参见[PAET89]的第一个附录。）

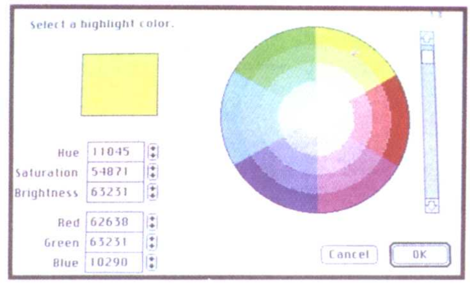
彩图16 加性基色。红色加绿色形成黄色，红色加蓝色形成品红色，绿色加蓝色形成青色，红色加绿色加蓝色形成白色。

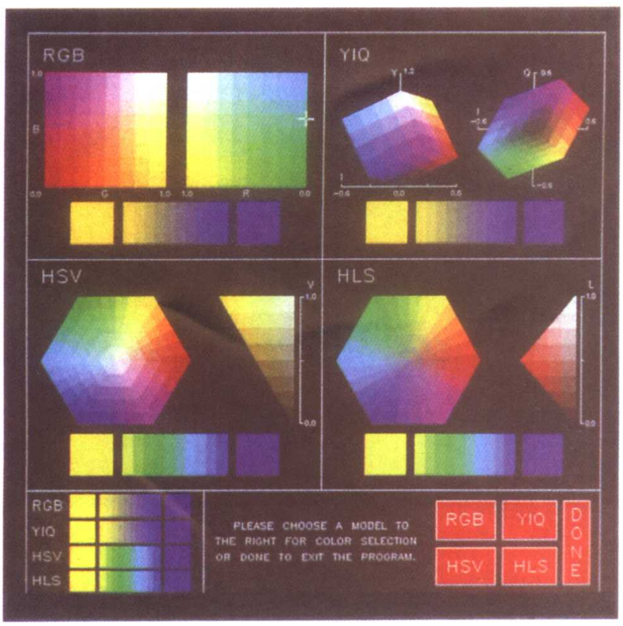




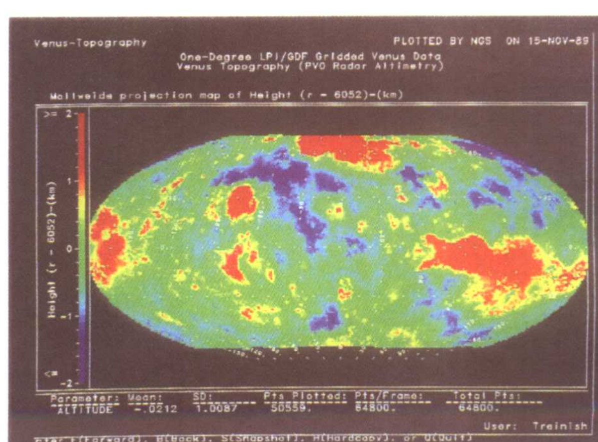
彩图17 减性基色。从白色中减去黄色和品红色形成红色，从白色中减去黄色和青色形成绿色，从白色中减去青色和品红色形成蓝色。

彩图18 Macintosh机器上使用的一种在HSV颜色空间内用于指定颜色的交互技术。色调和饱和度在圆形区域内给出、亮度在滑动条上给出。用户可移动圆形区域内的标记，改变滑动条上的标尺，也可键入新的HSV或RGB的数值。左上角正方形的彩色区域显示了当前的颜色及新的颜色。

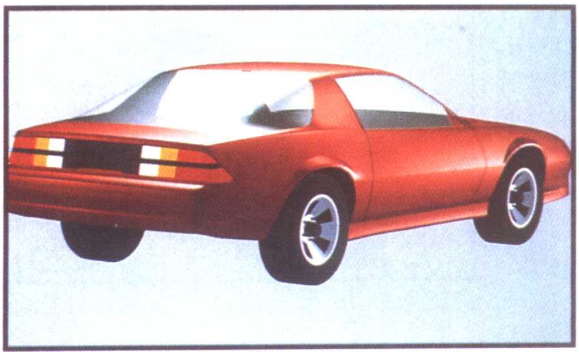




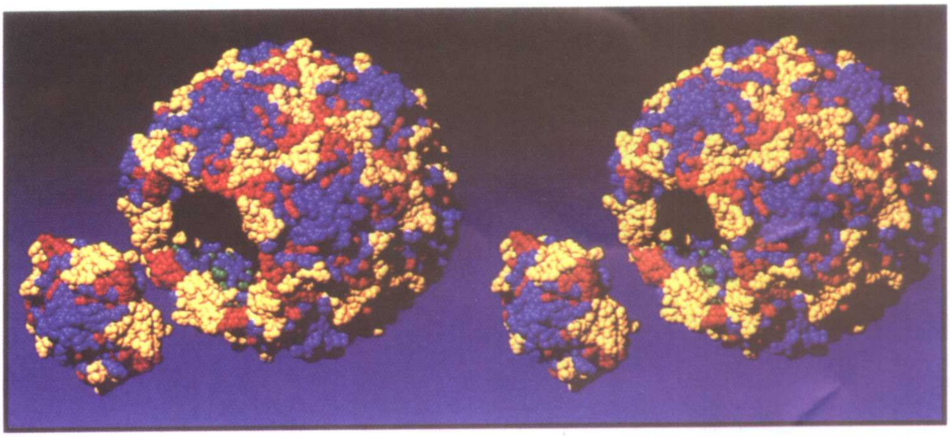
彩图19 可在四种不同颜色空间（RGB、YIQ、HSV及HLS）中，指定颜色并对颜色插值的一个交互程序。线性插值的起点和终点颜色通过在颜色空间的不同投影处定义来指定。每个颜色空间下面显示了插值，并且在左下角给出了比较。（经乔治·华盛顿大学Paul Charlton许可。）



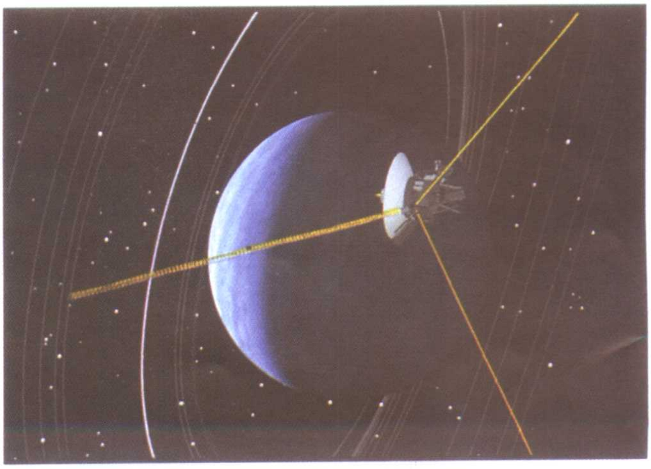
彩图20 一幅显示金星地形的伪彩色图像。左边的颜色标尺指示了在金星平均半径（6052 km）上下变化的高度（从－22 km到+2 km）。该图像是由NASA的绕金星“先锋”太空船经雷达测量获取数据，由月球及行星研究所对数据进行计算，并由美国国家空间科学数据中心的图形系统绘制的。（经NASA的Goddard空间飞行中心Lloyd Treinish许可。）



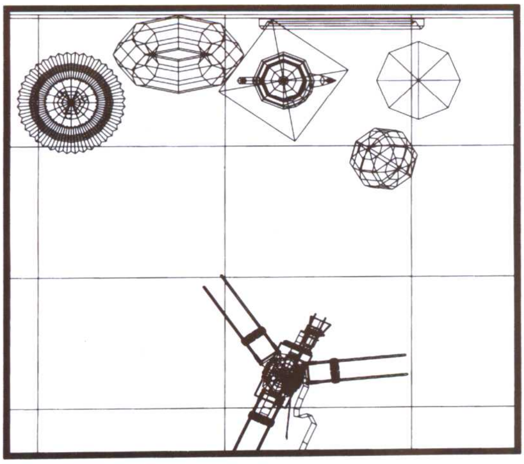
彩图21　按照Warn的光照控制照亮的雪佛兰Camaro。(经通用汽车公司研究实验室的David R.Warn许可。)



彩图22　一对立体的脑灰质炎病毒的衣壳，通过在每一个Alpha碳元素上放一个半径为0.5nm的小球而得到。移走了一个五节聚化物来显示其内部。经J.Hogle许可。(经David Goodsell 和 Arthur Olsen许可，Scripps临床研究院版权所有，1989。)

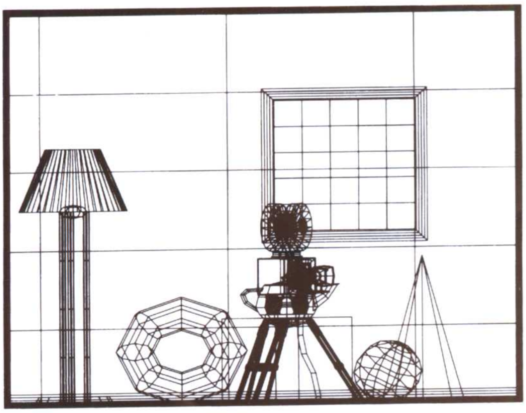


彩图23　模拟的带有光环和轨道的天王星的飞行。(经加利福尼亚理工学院的计算机图形实验室和喷气推动实验室的 Jim Blinn许可。)

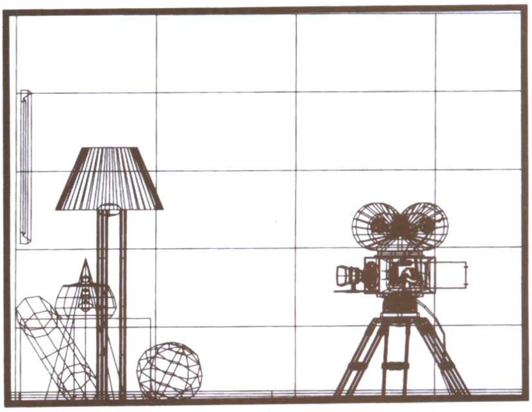


a)

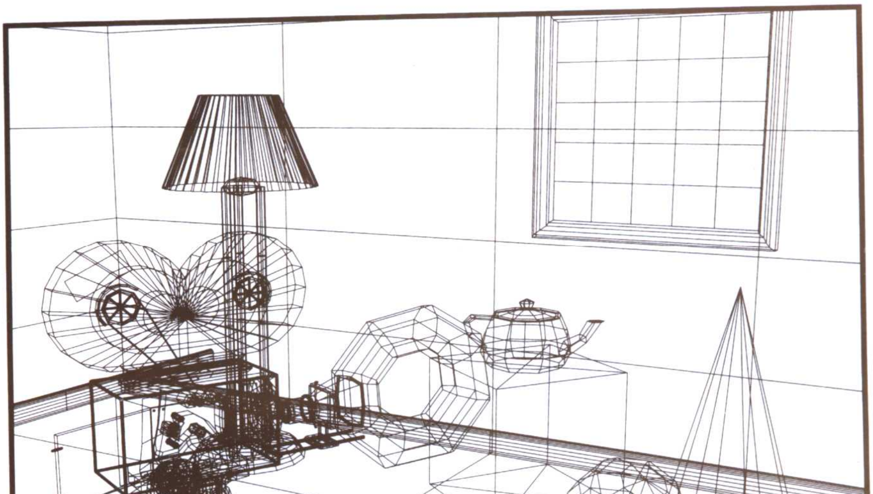
彩图24　摄影爱好者。带有电影摄像机的卧室场景。正交投影（6.2.2节和12.3.1节）。a) 俯视图，b) 正视图，c) 侧视图。多边形模型由样条面片产生。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）



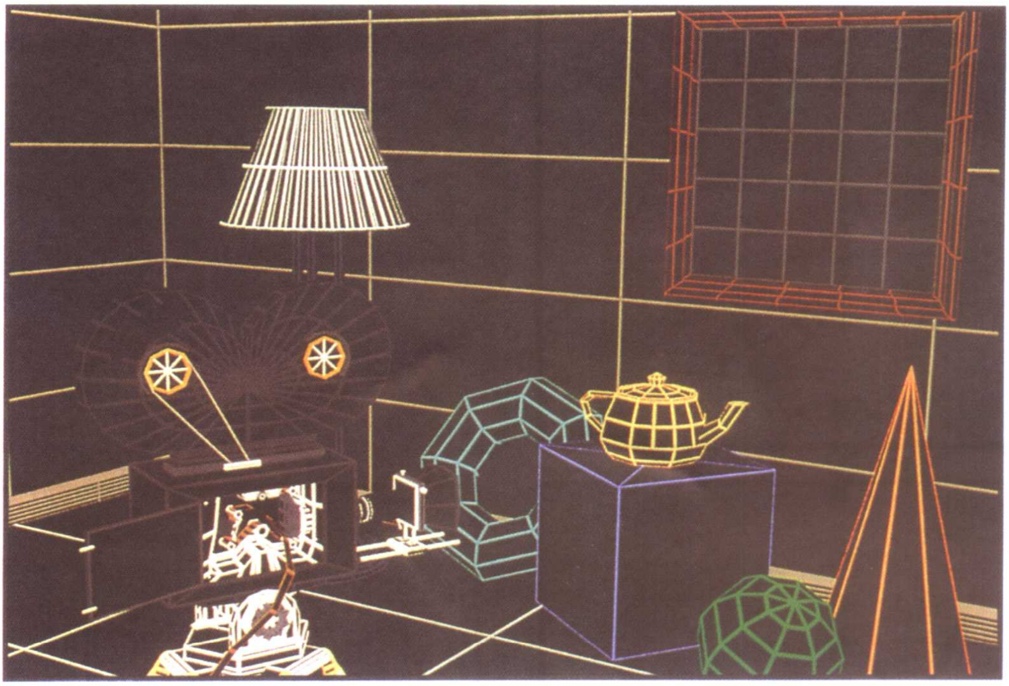
b)



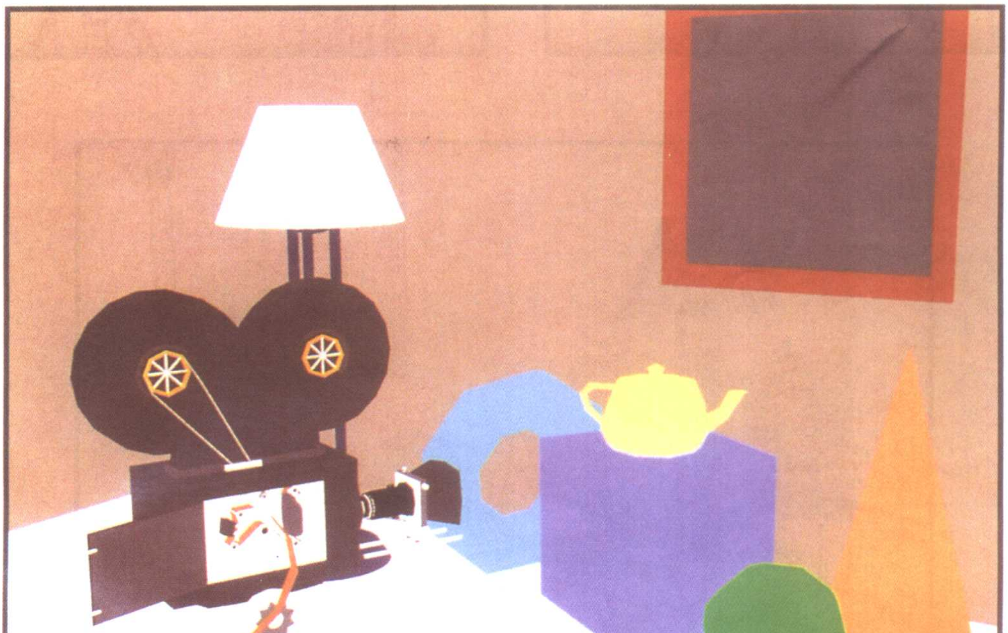
c)



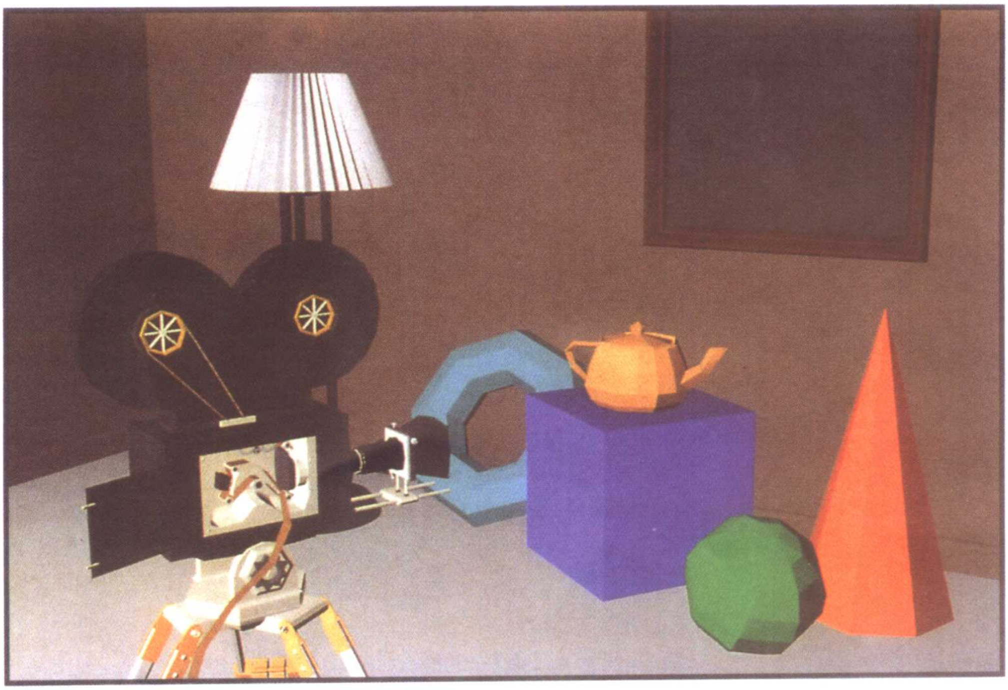
彩图25　摄影爱好者。透视投影（6.2.1节和12.3.2节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。）



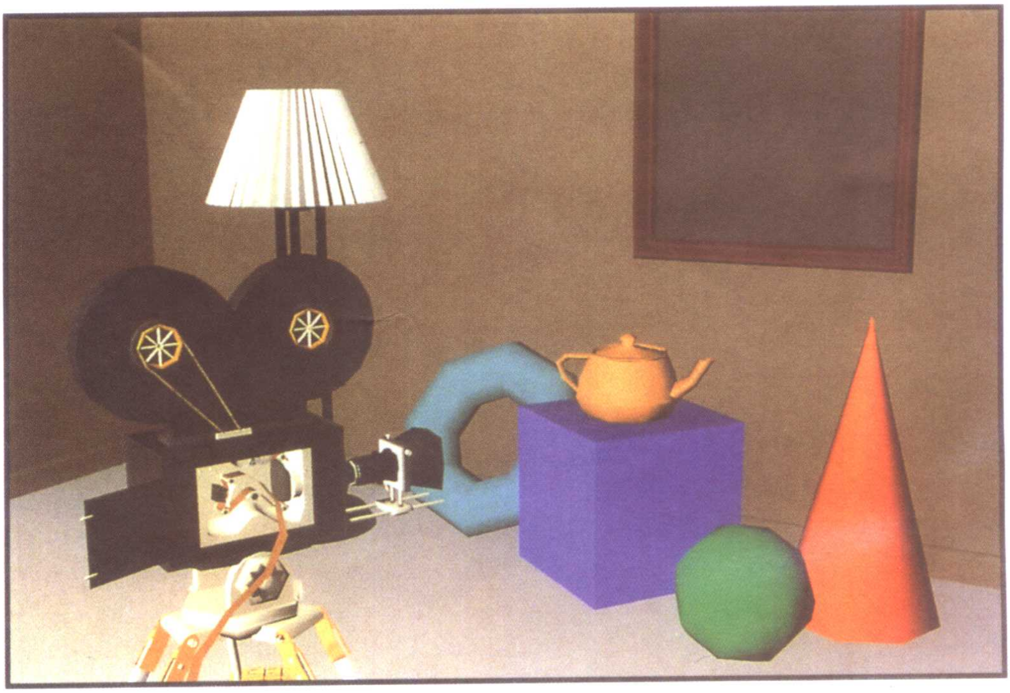
彩图 26　摄影爱好者。可见线判定（12.3.7节）。(Pixar公司版权所有，1990。由Thomas Williams和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。)



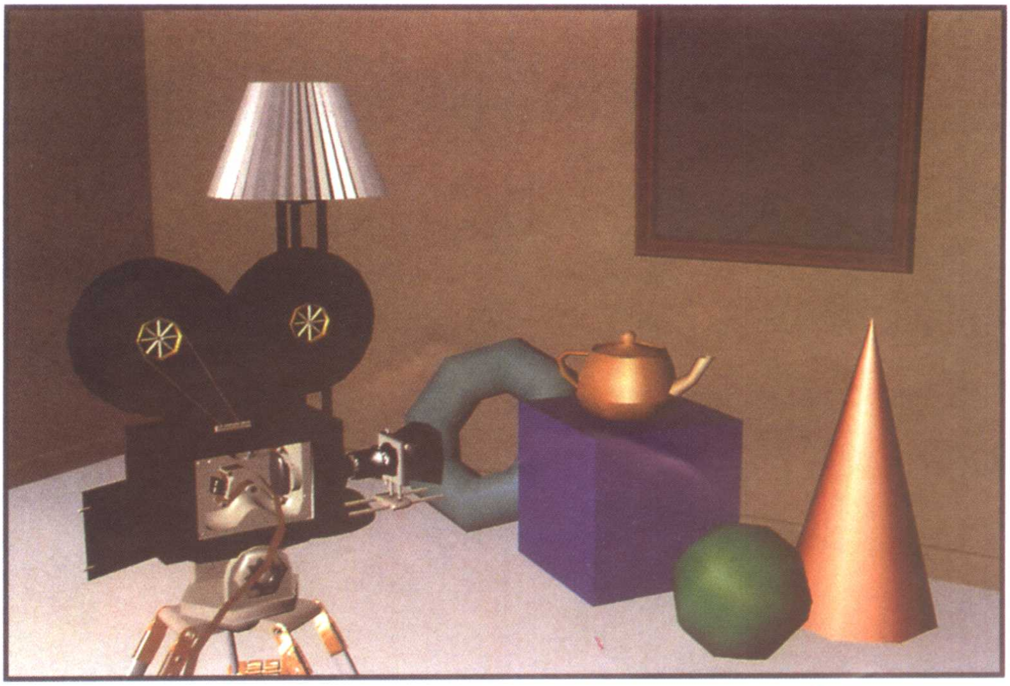
彩图 27　摄影爱好者。只带有环境光照的可见面判定（12.4.1节和14.1.1节）。(Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。)



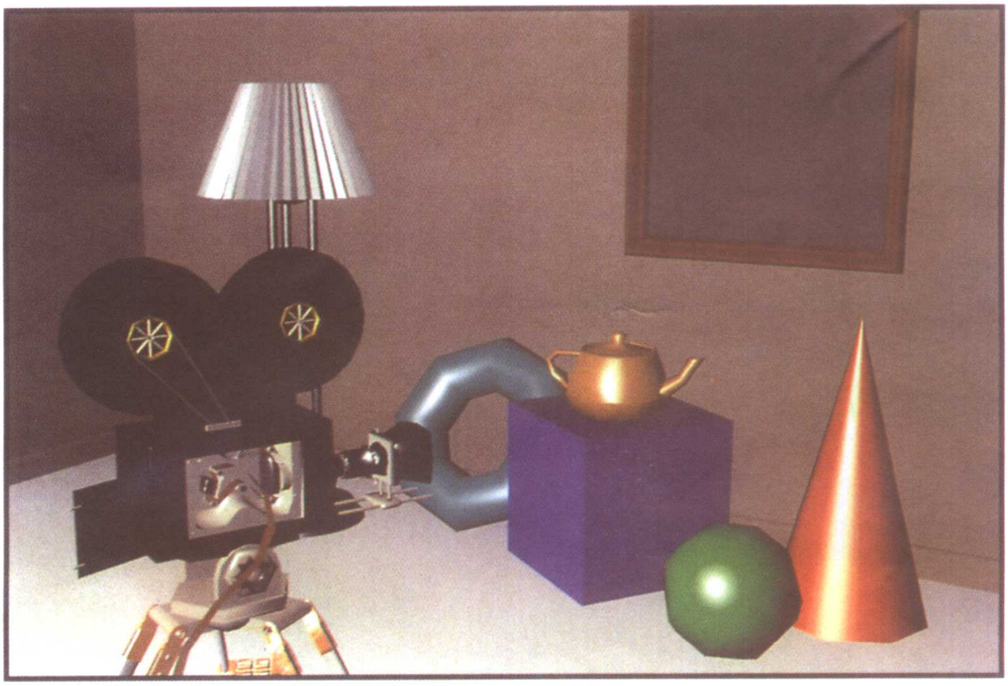
彩图28 摄影爱好者。带有漫反射的独立的经明暗处理的多边形（12.4.2节和14.2.3节）。(Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。)



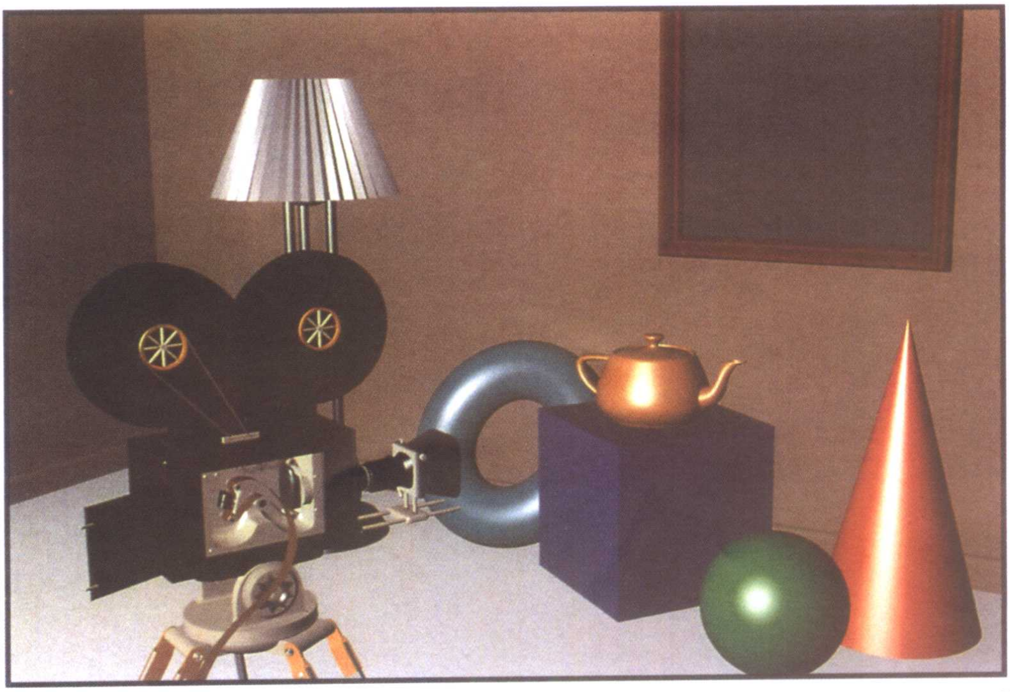
彩图29 摄影爱好者。带有漫反射的经Gouraud明暗处理的多边形（12.4.2节和14.2.3节）。(Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealisticRenderMan™软件绘制。)



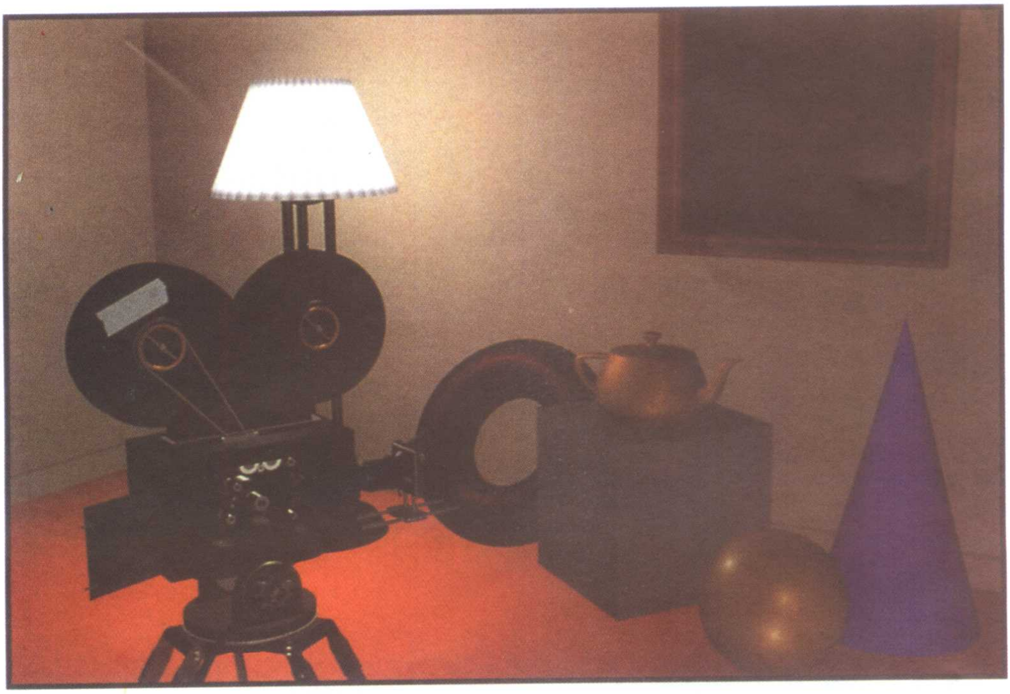
彩图30　摄影爱好者。带有镜面反射的经Gouraud明暗处理的多边形（12.4.4节和14.2.4节）。(Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用 Pixar的PhotoRealistic RenderMan^{TM}软件绘制。)



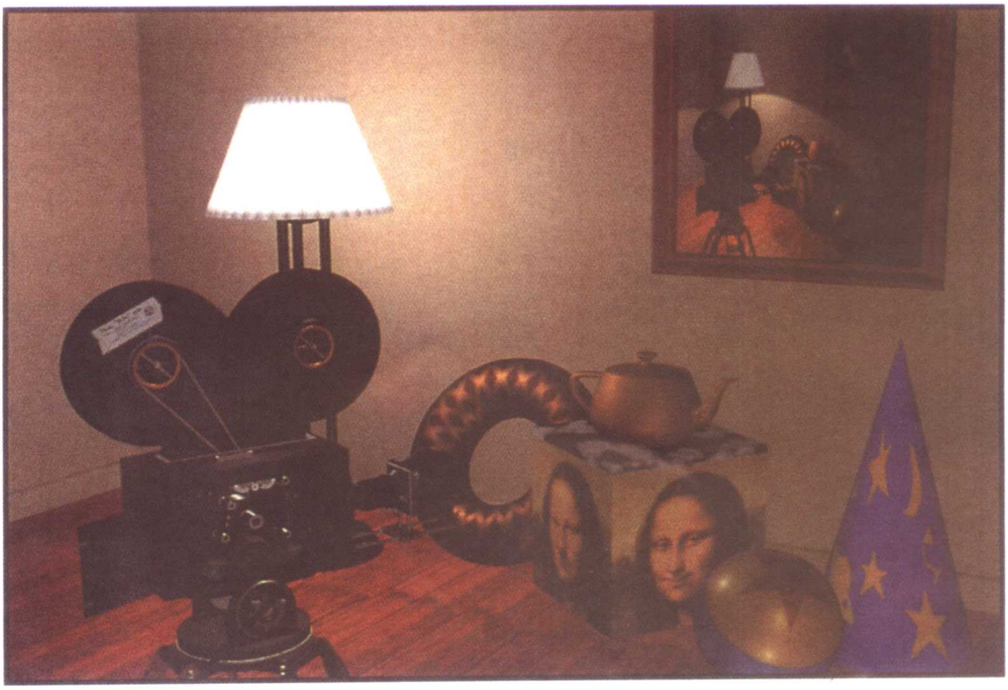
彩图31　摄影爱好者。带有镜面反射的经Phong明暗处理的多边形（12.4.4节和14.2.5节）。(Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用 Pixar的PhotoRealistic RenderMan^{TM}软件绘制。)



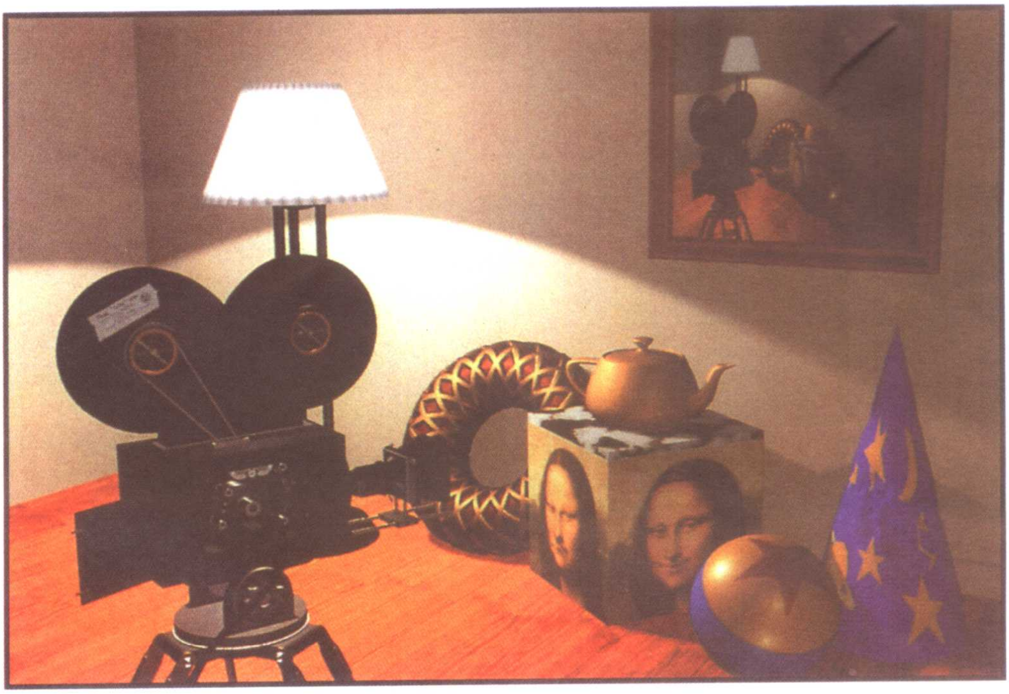
彩图32　摄影爱好者。带有镜面反射的曲面 （12.4.5节）。(Pixar公司版权所有，1990。由Thomas Williams和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。)



彩图33　摄影爱好者。改进的光照模型和多个光源（12.4.6节和14.1节）。(Pixar公司版权所有，1990。由ThomasWilliams 和H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。)



彩图34 摄影爱好者。纹理映射（12.4.7节和14.3.2节）。(Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。)

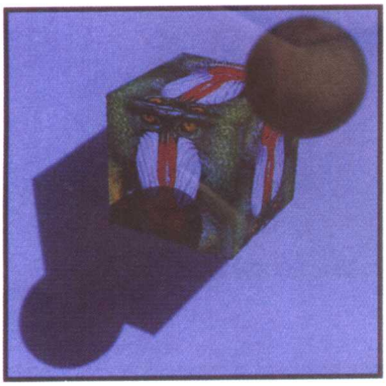


彩图35 摄影爱好者。位移映射（12.4.7节和14.3.4节）。(Pixar公司版权所有，1990。 由ThomasWilliams 和H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。)

彩图36 摄影爱好者。反射映射（12.4.9节）。(Pixar公司版权所有，1990。由Thomas Williams 和 H. B. Siegel使用 Pixar的Photo-Realistic RenderMan™软件绘制。)



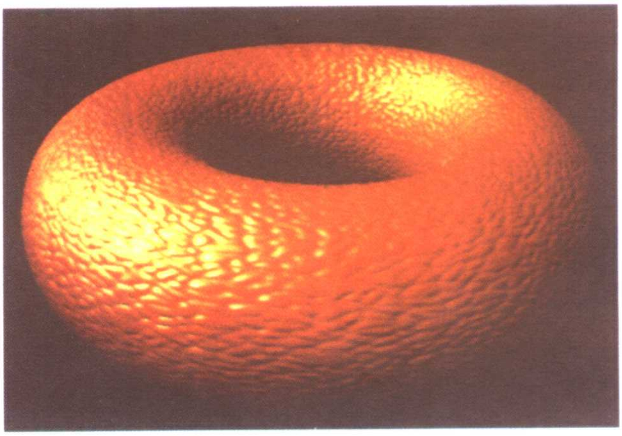
a)



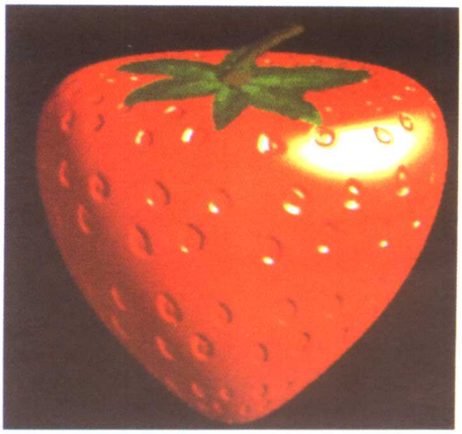
彩图37 通过后处理实现的景深（12.4.10节）。a) 聚焦在立方体上（550 mm），光圈是f/11。b) 聚焦在球体上(290 mm)，光圈是f/11。(经RPI的Michael Potmesil和Indranil Chakravarty许可。)



b)

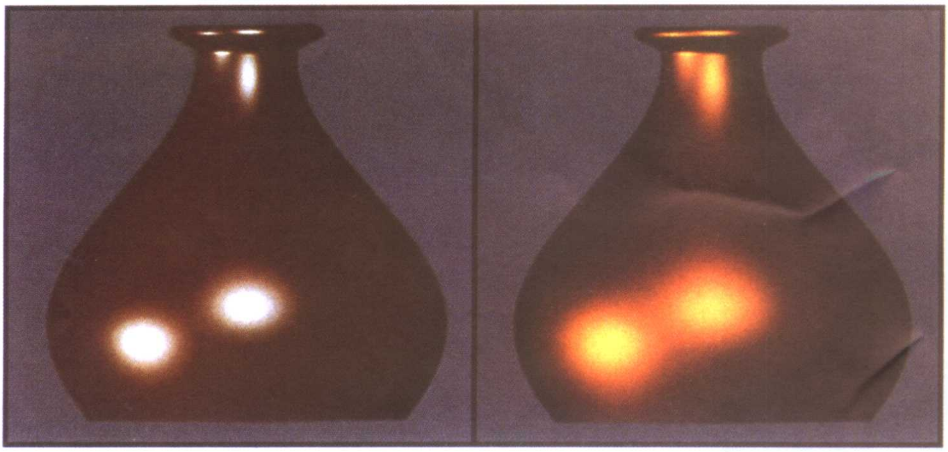


彩图38 用手工生成的凹凸函数映射的一个凹凸圆环面(14.3.3节)。(经犹他大学许可由Jim Blinn提供。)

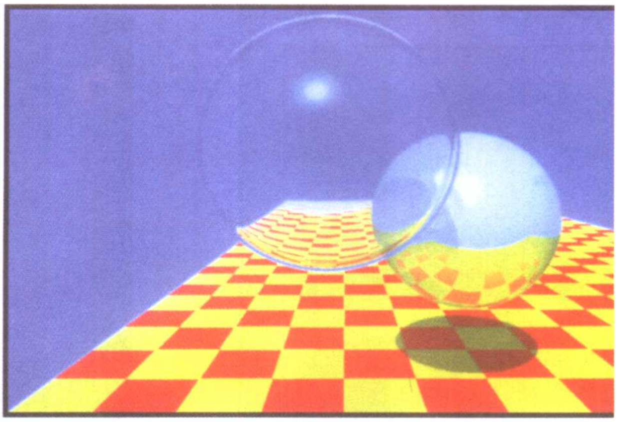


彩图39 用手工生成的凹凸函数映射的一个凹凸草莓（14.3.3节）。（经犹他大学许可由Jim Blinn提供。）

a)　　　　　　　　　　b)



彩图40 两个采用Cook-Torrance光照模型(14.1.7节）绘制的花瓶。两个花瓶都是被两个$I_{i1}=I_{i2}=$CIE 标准光源D6500 照射，其中$d\omega_{i1}=0.0001$，$d\omega_{i2}=0.0002$；$I_a=0.01I_{i1}$；ρ＝铜对法线入射方向的双向反射系数；$\rho_a=\pi\rho_d$；a)铜色的塑料：$k_s=0.1$，F＝乙烯树脂镜面的反射率；D＝Beckmann函数，其中参数$m=0.15$；$k_d=0.9$；b) 铜金属：$k_s=1.0$，F＝铜镜面的反射率；D＝Beckmann函数，其中参数m_1=0.4，$w_1=0.4$，$m_2=0.2$，$w_2=0.6$；$k_d=0.0$。（由康奈尔大学计算机图形学组的Robert Cook提供。）

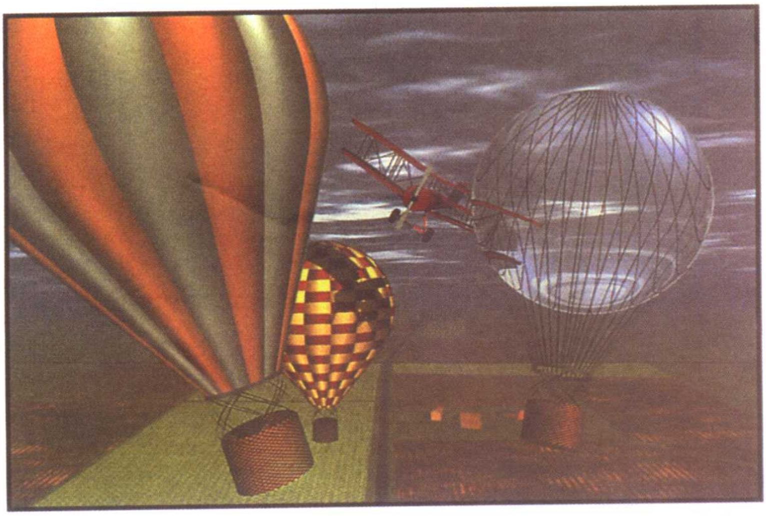


彩图41 球体和棋盘。一幅采用递归光线跟踪方法产生的早期图像（14.7节）。（经贝尔实验室的Turner Whitted许可。）

彩图42 光线跟踪产生的图像（14.7节）。a) 短电影《Quest》（1985）中的一个场景。（由Michael Sciulli、James Arvo和Melissa White制作。惠普公司版权所有。）b) “时尚的航空”。用函数修改了几乎所有像素的颜色、表面的法线和透明度。（经哥伦比亚大学的David Kurlander许可，1986。）

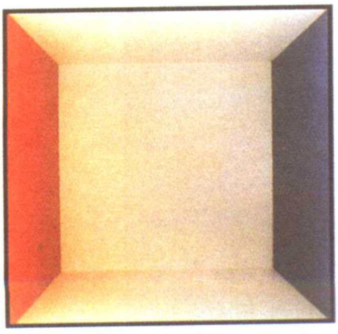


a)

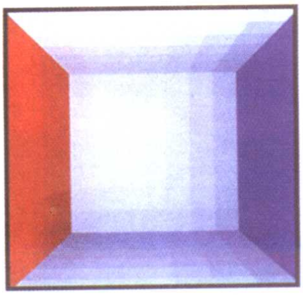


b)

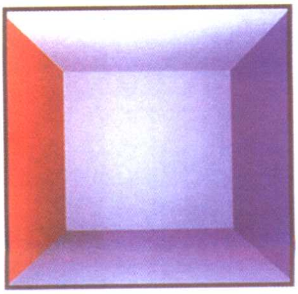
彩图43 辐射度（14.8.1节）。带有六个漫射墙壁的立方体（没有显示发白光的前墙）。a)真实立方体的照片，b) 每面墙由49个面片组成、每片用恒定明暗处理来绘制的模型；c)每面墙由49个面片组成、每片用插值的明暗处理来绘制的模型。（由康奈尔大学计算机图形学组的Cindy M.Goral、Kenneth E.Torrance、Donald P.Greenberg和Bennett Battaile提供，1984。）



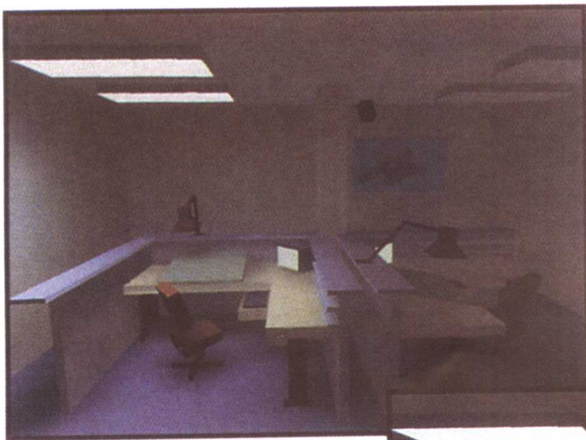
a)



b)

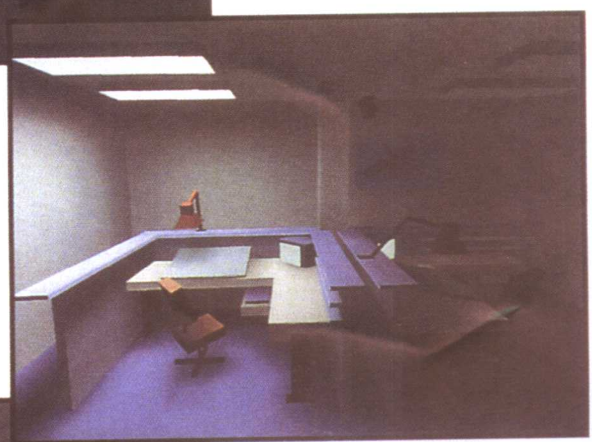


c)

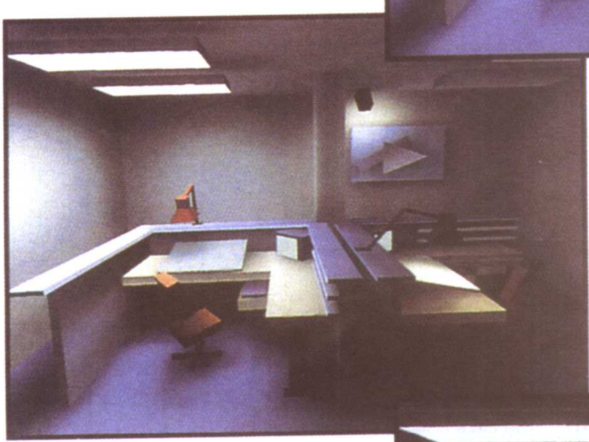


a)

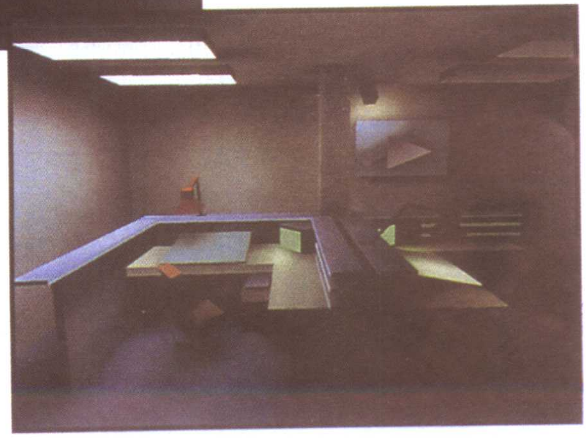
彩图44 采用逐步求精的半立方体辐射度算法（14.8.3节）绘制的办公室；包含500个面片，7000个子面片。加入了估计的泛光辐射度，在一台HP 9000 825 SRX型计算机上，每一次迭代的计算和显示大约需要15秒。a)1次迭代，b)2次迭代，c)24次迭代，d)100次迭代。（由康奈尔大学计算机图形学组的Shenchang Eric Chen、Michael F.Cohen、John R.Wallace和Donald P.Greenberg提供，1988。）



b)



c)



d)

目 录

出版者的话
专家指导委员会
译序
译者简介
前言
作者简介
《计算机图形学原理及实践——C语言描述（第2版）》前言
第1章 计算机图形学简介 …… 1
1.1 计算机图形学的几种用途 …… 1
1.2 计算机图形学的简单回顾 …… 4
1.2.1 输出技术 …… 5
1.2.2 输入技术 …… 8
1.2.3 软件的可移植性与图形标准 …… 8
1.3 交互式图形学的优点 …… 9
1.4 交互式图形学的概念框架 …… 10
1.4.1 应用建模 …… 10
1.4.2 模型的显示 …… 11
1.4.3 交互处理 …… 11
小结 …… 12
习题 …… 13
第2章 简单光栅图形软件包(SRGP)的编程 …… 15
2.1 用SRGP画图 …… 15
2.1.1 图形图元的规格 …… 15
2.1.2 属性 …… 19
2.1.3 填充图元及其属性 …… 20
2.1.4 存储和恢复属性 …… 23
2.1.5 文本 …… 23
2.2 基本交互处理 …… 24
2.2.1 人的因素 …… 24
2.2.2 逻辑输入设备 …… 25
2.2.3 采样与事件驱动处理 …… 26
2.2.4 采样模式 …… 27
2.2.5 事件模式 …… 28
2.2.6 交互处理中的关联拾取 …… 32
2.2.7 设置设备度量和属性 …… 33
2.3 光栅图形特性 …… 34
2.3.1 画布 …… 34
2.3.2 矩形框的裁剪 …… 36
2.3.3 SRGP_copyPixel操作 …… 36
2.3.4 写模式或RasterOp …… 38
2.4 SRGP的局限性 …… 41
2.4.1 应用程序坐标系统 …… 41
2.4.2 为了重新定义存储图元 …… 41
小结 …… 42
习题 …… 43
程序设计项目 …… 44
第3章 二维图元的基本光栅图形学算法 …… 45
3.1 概述 …… 45
3.1.1 显示系统体系结构的含义 …… 45
3.1.2 软件中的输出流水线 …… 48
3.2 直线的扫描转换 …… 48
3.2.1 基本增量算法 …… 49
3.2.2 中点线算法 …… 50
3.2.3 补充要点 …… 54
3.3 圆的扫描转换 …… 56
3.3.1 八方向对称性 …… 56
3.3.2 中点圆算法 …… 57
3.4 填充矩形 …… 59
3.5 填充多边形 …… 60
3.5.1 水平边 …… 62
3.5.2 狭长条 …… 62
3.5.3 边相关性和扫描线算法 …… 63
3.6 图案填充 …… 65
3.6.1 使用扫描转换的图案填充 …… 65
3.6.2 不用重复扫描转换的图案填充 …… 66
3.7 宽图元 …… 68
3.7.1 复制像素 …… 68
3.7.2 移动画笔 …… 69
3.8 光栅空间的裁剪操作 …… 70
3.9 线段裁剪 …… 70
3.9.1 裁剪端点 …… 71

3.9.2 利用求解联立方程组的线段裁剪……71
3.9.3 Cohen-Sutherland 线裁剪算法 ………72
3.9.4 参数化的线裁剪算法…………………75
3.10 圆的裁剪………………………………78
3.11 多边形裁剪……………………………78
3.12 生成字符………………………………81
3.12.1 定义和裁剪字符 ……………………81
3.12.2 一种文本输出图元的实现 …………83
3.13 SRGP_copyPixel ………………………84
3.14 反走样…………………………………84
3.14.1 增加分辨率 …………………………84
3.14.2 未加权的区域采样 …………………85
3.14.3 加权区域采样 ………………………86
3.15 高级主题………………………………88
小结 ……………………………………………89
习题 ……………………………………………89
第4章 图形硬件 ………………………………91
4.1 硬拷贝技术 ……………………………92
4.2 显示技术 ………………………………94
4.3 光栅扫描显示系统 ……………………98
4.3.1 简单的光栅显示系统…………………99
4.3.2 具有外围显示处理器的光栅显示系统 ……………………………………101
4.3.3 显示处理器的附加功能 ……………103
4.3.4 具有集成显示处理器的光栅显示系统 ……………………………………104
4.4 视频控制器……………………………105
4.5 用于操作者交互的输入设备…………106
4.5.1 定位设备 ……………………………106
4.5.2 键盘设备 ……………………………108
4.5.3 定值设备 ……………………………108
4.5.4 选择设备 ……………………………108
4.6 图像扫描仪……………………………109
习题……………………………………………109
第5章 几何变换 ………………………………111
5.1 数学基础………………………………111
5.1.1 向量及其性质 ………………………111
5.1.2 向量点积 ……………………………112
5.1.3 点积的性质 …………………………113
5.1.4 矩阵 …………………………………113
5.1.5 矩阵乘法 ……………………………114
5.1.6 行列式 ………………………………114
5.1.7 矩阵的转置 …………………………115
5.1.8 矩阵的逆 ……………………………115
5.2 二维变换………………………………116
5.3 齐次坐标和二维变换的矩阵表示 ……118
5.4 二维变换的合成………………………121
5.5 窗口到视口的变换……………………122
5.6 效率……………………………………124
5.7 三维变换的矩阵表示…………………125
5.8 三维变换的合成………………………127
5.9 坐标系的变换…………………………131
习题……………………………………………133
第6章 三维空间的观察 ………………………135
6.1 人造照相机及三维观察步骤…………135
6.2 投影……………………………………136
6.2.1 透视投影 ……………………………137
6.2.2 平行投影 ……………………………138
6.3 指定一个任意的三维视图……………140
6.4 三维观察的例子………………………142
6.4.1 透视投影 ……………………………143
6.4.2 平行投影 ……………………………146
6.4.3 有限的视见体 ………………………147
6.5 平面几何投影的数学…………………147
6.6 实现平面几何投影……………………150
6.6.1 平行投影 ……………………………151
6.6.2 透视投影 ……………………………155
6.6.3 用三维规范视见体进行裁剪 ………158
6.6.4 在齐次坐标中裁剪 …………………159
6.6.5 映射到一个视口 ……………………161
6.6.6 实现小结 ……………………………162
6.7 坐标系统………………………………163
习题……………………………………………164
第7章 对象的层次结构和简单的PHIGS系统 ……………………………………167
7.1 几何造型………………………………168
7.1.1 几何模型 ……………………………169
7.1.2 几何模型中的层次 …………………169
7.1.3 模型、应用程序和图形系统间的关系 ………………………………171
7.2 保留模式图形包的特点………………172
7.2.1 中央结构存储库及其优点 …………172

7.2.2 保留模式软件包的局限性 …………173
7.3 定义和显示结构……………………………173
7.3.1 打开和关闭结构 ……………………173
7.3.2 定义输出图元及其属性 ……………174
7.3.3 提交结构进行显示遍历 ……………176
7.3.4 观察 ……………………………………177
7.3.5 通过窗口管理共享屏幕的图像应用…179
7.4 模型变换……………………………………180
7.5 层次式结构网络……………………………183
7.5.1 两层层次结构 ………………………183
7.5.2 简单的三层层次结构 ………………184
7.5.3 自底向上构造机器人 ………………185
7.5.4 交互式造型程序 ……………………187
7.6 显示遍历中的矩阵合成……………………187
7.7 层次结构中外观属性的处理………………190
7.7.1 继承法则 ……………………………190
7.7.2 SPHIGS的属性及文字不受变换影响 ……………………………………191
7.8 屏幕的更新和绘制模式……………………192
7.9 用于动态效果的结构网络编辑……………193
7.9.1 利用索引和标记访问元素 …………193
7.9.2 内部结构的编辑操作 ………………194
7.9.3 改进编辑方法的一些实例块 ………195
7.9.4 如何控制屏幕图像的自动再生 ……196
7.10 交互 ………………………………………196
7.10.1 定位器……………………………………197
7.10.2 关联拾取…………………………………197
7.11 高级问题 …………………………………202
7.11.1 其他输出特性…………………………202
7.11.2 实现问题………………………………202
7.11.3 层次模型的优化显示…………………203
7.11.4 PHIGS中层次模型的局限性…………204
7.11.5 层次建模的其他形式…………………204
7.11.6 其他（工业）标准……………………204
小结…………………………………………………205
习题…………………………………………………206
第8章 输入设备、交互技术与交互任务…207
8.1 交互硬件……………………………………207
8.1.1 定位设备 ……………………………208
8.1.2 键盘设备 ……………………………209
8.1.3 定值设备 ……………………………209
8.1.4 选择设备 ……………………………209
8.1.5 其他设备 ……………………………209
8.1.6 三维交互设备 ………………………210
8.2 基本交互任务………………………………211
8.2.1 定位交互任务 ………………………211
8.2.2 选择交互任务——大小可变的选项集合 ……………………………………212
8.2.3 选择交互任务——相对固定大小的选项集合 ……………………………214
8.2.4 文本交互任务 ………………………216
8.2.5 定量交互任务 ………………………216
8.2.6 三维交互任务 ………………………216
8.3 复合交互任务………………………………218
8.3.1 对话框 ………………………………218
8.3.2 构造技术 ……………………………218
8.3.3 动态操纵 ……………………………219
8.4 交互技术工具箱……………………………221
小结…………………………………………………221
习题…………………………………………………221
第9章 曲线与曲面的表示 ……………………223
9.1 多边形网格…………………………………224
9.1.1 多边形网格的表示 …………………224
9.1.2 平面方程 ……………………………226
9.2 三次参数曲线………………………………227
9.2.1 基本特性 ……………………………228
9.2.2 Hermite曲线…………………………230
9.2.3 Bézier曲线 …………………………233
9.2.4 均匀非有理B样条曲线 ……………238
9.2.5 非均匀非有理B样条曲线 …………241
9.2.6 非均匀有理三次多项式曲线段 ……242
9.2.7 用数字化点来拟合曲线 ……………242
9.2.8 三次曲线的比较 ……………………243
9.3 双三次参数曲面……………………………243
9.3.1 Hermite曲面…………………………244
9.3.2 Bézier曲面 …………………………245
9.3.3 B样条曲面 …………………………246
9.3.4 曲面的法线 …………………………246
9.3.5 双三次曲面的显示 …………………247
9.4 二次曲面……………………………………248
9.5 专用的造型技术……………………………249

9.5.1 分形模型 ……249
9.5.2 基于文法的模型 ……252
小结……254
习题……254
第10章 实体造型 ……257
10.1 实体表示 ……257
10.2 正则布尔集合运算 ……258
10.3 基本实体举例法 ……260
10.4 扫掠表示法 ……261
10.5 边界表示法 ……262
10.5.1 多面体和欧拉公式……262
10.5.2 布尔集合运算……264
10.6 空间划分表示法 ……264
10.6.1 单元分解法……264
10.6.2 空间位置枚举法……265
10.6.3 八叉树表示法……265
10.6.4 二元空间划分树……268
10.7 构造实体几何 ……269
10.8 各种表示法的比较 ……270
10.9 实体造型的用户界面 ……271
小结……272
习题……272
第11章 消色差光与彩色光……273
11.1 消色差光 ……273
11.1.1 选择亮度值……273
11.1.2 半色调逼近……275
11.2 彩色 ……277
11.2.1 心理物理学……277
11.2.2 CIE色度图 ……279
11.3 用于光栅图形的颜色模型 ……281
11.3.1 RGB颜色模型 ……282
11.3.2 CMY颜色模型 ……282
11.3.3 YIQ颜色模型 ……283
11.3.4 HSV颜色模型……284
11.3.5 颜色的交互指定……286
11.3.6 在颜色空间中进行插值……287
11.4 在计算机图形学中应用颜色 ……288
小结……289
习题……290
第12章 可视图像真实感的探讨 ……291
12.1 为什么讨论真实感 ……291
12.2 基本的困难 ……292
12.3 线条图的绘制技术 ……293
12.3.1 多正交视图……293
12.3.2 透视投影……294
12.3.3 深度提示……294
12.3.4 深度裁剪……294
12.3.5 纹理……295
12.3.6 颜色……295
12.3.7 可见线的判定……295
12.4 明暗图像的绘制技术 ……295
12.4.1 可见面的判定……295
12.4.2 光照和明暗处理……295
12.4.3 插值明暗处理……296
12.4.4 材质属性……296
12.4.5 曲面造型……296
12.4.6 改进光照和明暗效果……296
12.4.7 纹理……296
12.4.8 阴影……296
12.4.9 透明性和反射……297
12.4.10 改进的相机模型 ……297
12.5 改进的物体模型 ……297
12.6 动力学与动画 ……297
12.6.1 运动的价值……297
12.6.2 动画……298
12.7 体视学 ……300
12.8 改进的显示技术 ……300
12.9 与其他感官的交互 ……300
小结……301
习题……301
第13章 可见面的判定 ……303
13.1 有效可见面判定算法的技术 ……304
13.1.1 相关性……304
13.1.2 透视变换……305
13.1.3 范围与包围体……306
13.1.4 背面消除……307
13.1.5 空间划分……308
13.1.6 层次结构……308
13.2 z缓存算法……309
13.3 扫描线算法 ……311
13.4 可见面光线跟踪 ……314

13.4.1 相交计算……315
13.4.2 可见面光线跟踪算法的效率……316
13.5 其他方法……317
13.5.1 列表优先级算法……317
13.5.2 区域细分算法……320
13.5.3 曲面算法……322
小结……322
习题……323
第14章 光照和明暗处理……325
14.1 光照模型……325
14.1.1 环境光……325
14.1.2 漫反射……326
14.1.3 大气衰减……329
14.1.4 镜面反射……329
14.1.5 点光源模型的改进……331
14.1.6 多光源……333
14.1.7 基于物理的光照模型……333
14.2 多边形的明暗处理模型……335
14.2.1 恒定明暗处理……335
14.2.2 插值明暗处理……335
14.2.3 多边形网格的明暗处理……335
14.2.4 Gouraud 明暗处理技术……336
14.2.5 Phong明暗处理技术……337
14.2.6 插值明暗处理中的问题……338
14.3 曲面细节……339
14.3.1 曲面细节多边形……339
14.3.2 纹理映射……339
14.3.3 凹凸映射……340
14.3.4 其他方法……341
14.4 阴影……341
14.4.1 扫描线生成阴影算法……341
14.4.2 阴影体……342
14.5 透明性……343
14.5.1 无折射的透明性……343
14.5.2 折射透明性……345
14.6 全局光照算法……346
14.7 递归光线跟踪……347
14.8 辐射度方法……350
14.8.1 辐射度方程……350
14.8.2 计算形状因子……352
14.8.3 逐步求精算法……353
14.9 绘制流水线……354
14.9.1 局部光照绘制流水线……354
14.9.2 全局光照绘制流水线……355
14.9.3 逐步求精方法……356
小结……356
习题……357
参考文献……359
索引……377

第1章 计算机图形学简介

欢迎进入计算机图形学这一计算机科学中最激动人心的领域。的确，计算机图形学是计算机科学的一个分支，但是它的吸引力已是其他相关专门领域所望尘莫及的。在其短暂的成长过程中，计算机图形学已经吸引了世界上最有创造性的人们进入了它的领域。他们来自各行各业：艺术、科学、音乐、舞蹈、电影制作，以及许多其他行业。仅举数例，迪斯尼的动画师在《美女与野兽》中，用计算机图形技术赋予舞厅场景以特殊的魅力。舞蹈指导Merce Cunningham使用计算机图形技术创作了拉班舞谱（labanotation），一种让舞蹈家学习舞步的手稿。受尊敬的传统媒体艺术家David Em，在他的工作中现在广泛地使用了计算机图形技术。事实上，计算机图形学能通过对其应用的考虑很好地表达它的刺激性和多样性，让我们更详尽地看看它的几种应用。

1.1 计算机图形学的几种用途

今天，计算机图形已被用于工业、商业、政府、教育、娱乐等各个方面。最近，还进入了家庭。其应用不胜枚举，而且，随着图形能力成为日常商品，其应用还在不断增加。让我们看一看其中的一些领域。

- **用户界面**：您可以不明白什么是计算机图形学，但是您可能已经使用了它。如果您已经在一台Macintosh计算机或一台运行Windows 3.1的IBM兼容个人计算机上工作，那么您实际上已经是一位经验丰富的图形用户。毕竟运行在个人计算机及工作站上的大多数应用程序具有用户界面（类似于图1-1所示的界面），这些用户界面依赖于桌面窗口系统来管理多个同时发生的活动，依赖于一种指示能力让用户来选择菜单项、图标和屏幕上的对象。字处理、电子表格及桌面出版程序均是具有这样用户界面技术优势的典型应用程序。正像我们以后要了解的，计算机图形学在用户界面的输入/输出两方面功能中扮演了一个主要的角色。 1
- **商业与科技中的（交互）绘图**：当今图形的另一个极为广泛的应用也许是绘制二维和三维的数学、物理或经济函数的图；柱状图、条状图或饼图，任务调度图，库存和生产图等。所有这些图都是用来明确而简洁地表示由数据抽取出来的趋势和模式，从而将复杂的现象分类，做出有见识的决定。图1-2给出了商业数据的3D绘图的典型例子。
- **制图学**：计算机图形被用来从测量数据生成地理或其他自然现象的准确和图解的表述。其范例包括地图、地形图、用于钻井和开采的探矿图、航海图、气象图、等高线图和人口密度图。图1-3展示了一张地图，该图的制作是用来说明1989年Exxon公司在美国阿拉斯加州南部港市瓦尔迪兹（Valdez）发生原油溢出灾难的一些情况。该图是主题图（thematic）的一个例子，它的数据值（例如："溢出原油的流动"）覆盖在地图的本底上。
- **医学**：计算机图形学在诸如诊断医学和外科手术规划等领域正在扮演一个不断增强的角色。对于后者，手术时使用图形技术作为导向器械的辅助，并精确地确定患病组织应该切除的位置。图1-4展示了计算机图形学在诊断医学中的一个应用。我们可看到那里有人脑的三幅由核磁共振（MRI）派生的图像。从一系列并行扫描得到的如图1-4a所示的图片，诊断专家能够合成为三维的人脑表示，如图1-4b及图1-4c所示。用户可交互地操作

此模型来揭示人脑情况的详细信息。另外，图1-5展示的计算机图形学的一个医学应用，令人特别激动。这是一种称为“光学活组织切片检查”的技术，我们可看到从激光照射活体组织而得到的数据所绘制的三维图。图中，正常的（图1-5a）和癌变的（图1-5b）犬肝脏显示了十分不同的光学信号，它为非手术诊断提供了希望。

- **计算机辅助制图和设计**：在计算机辅助设计（CAD）中，交互式图形用来设计各种机械的、电气的、机电一体的和电子设备中的元件和系统，包括了诸如建筑物、汽车车体、飞机机身和轮船船体、大规模集成电路（VLSI）芯片、电话与计算机网络的结构。通常，我们强调的是与设计当中的元件或系统的计算机模型交互，但是，有些时候，用户并不希望画出精确的零件图和装配图，而只是在线画草图或建筑蓝图。图1-6给出了建筑CAD的例子，它是一个发电站的图。
- **多媒体系统**：计算机图形学在快速发展的多媒体系统领域扮演了一个重要的角色。正像“多媒体”的名称的含义，多媒体包含了多于一种通信的媒体。按惯例，我们可以在这样的系统中看到正文、图形及声音，然后还有许多其他媒体[PHIL91]。图1-7展示了在教育中是如何使用非常规多媒体组件的。该图是从一本假想的计算机图形学多媒体教科书中派生而来的[PHIL92]。图1-7a是画一个2D线段的过程（我们将在第3章中讨论该代码的含义）。当该代码出现在多媒体页面上时，它是实况转播的；也即，当读者指点它时，则同时执行它。图1-7b展示了一个算法实验面板，它让读者试验不同的终点条件，同时观察算法的动作。
- **科学计算可视化和环境的模拟与动画**：对于科学与工程可视化而言，计算机生成的动画影片和真实对象或模拟对象的时变行为的显示正变得日益流行（见彩图1）。我们可以用它们来学习抽象的数学实体，或研究许多现象的数学模型，例如，液体流动、相对论、核反应与化学反应、生理系统与组织功能以及机械结构在各种负荷下的变形等。另一个先进技术领域是电影中的一流特技（见彩图2和彩图3）。复杂的机制可用于物体的造型和光与阴影的表现。

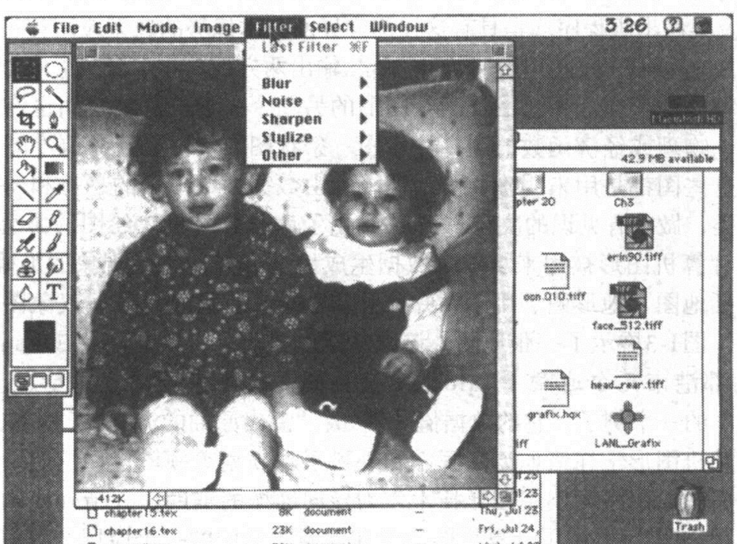


图1-1 一个Macintosh计算机的屏幕，显示了窗口、菜单和桌面图标

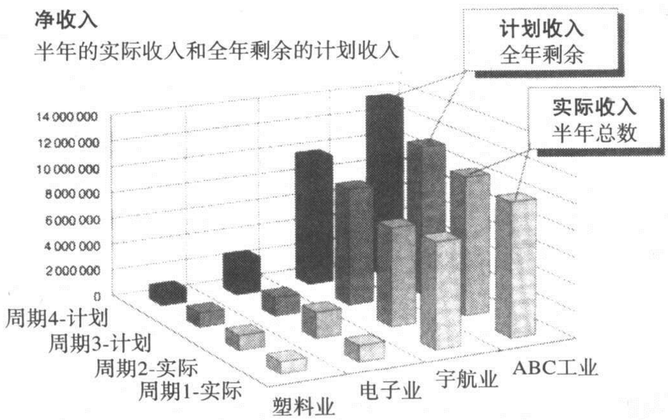


图1-2 显示成三维柱状图的商业数据

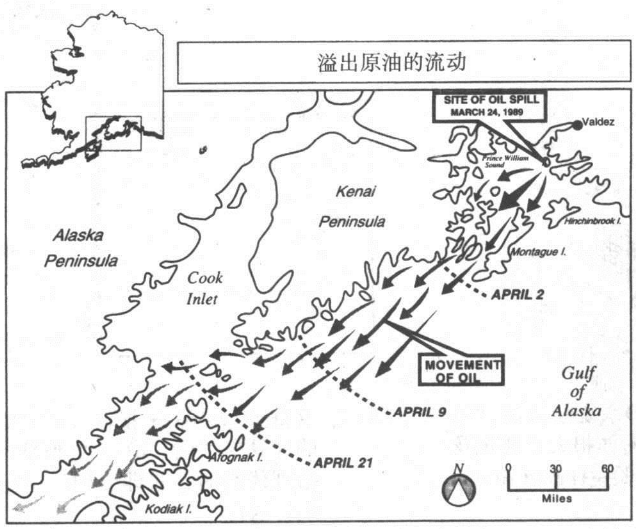


图1-3 使用地图的本底来描绘地理上有关联的数据。（经Simon Fraser 大学的Tom Poiker许可。）

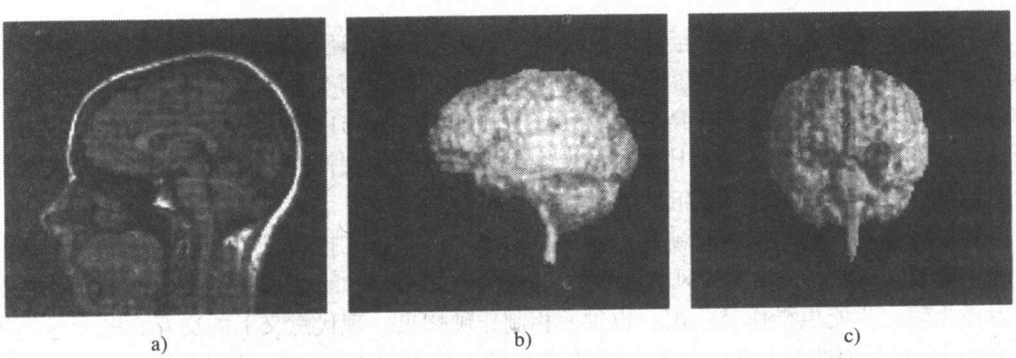
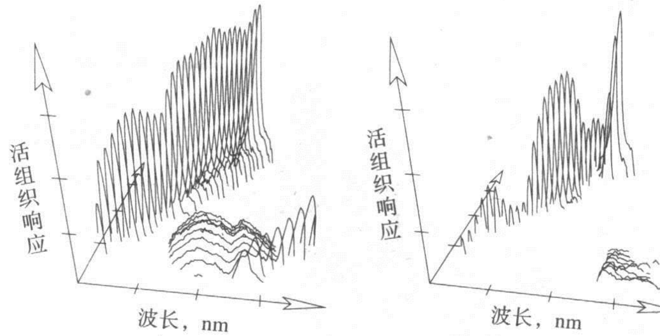


图1-4 从核磁共振扫描图像构造而成的人脑3D表示。由a) 所示的一系列这类图像经处理而产生的3D景象，b) 为侧视图，c) 为后视图。（经Los Alamos国家实验室的 John George许可。）



a) b)

图1-5 一个光学活组织切片检查结果。a) 为正常肝组织的信号，b) 为癌变肝组织的信号。（经Los Alamos国家实验室的 Thomas Loree许可。）

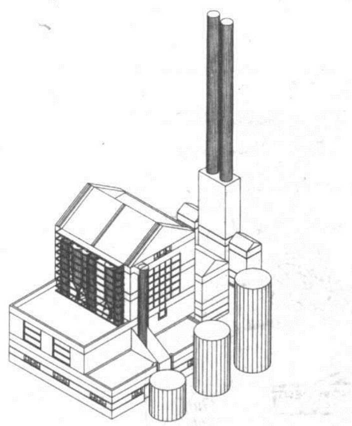
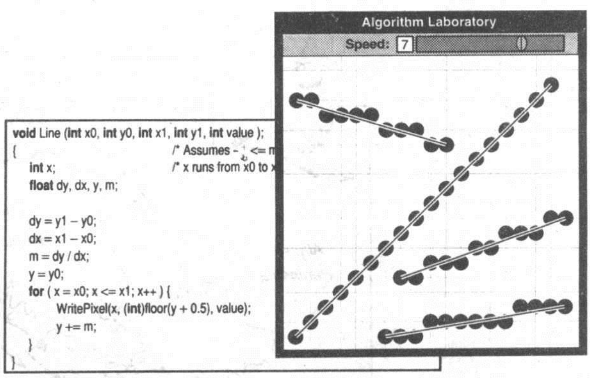


图1-6 由建筑CAD系统绘制的发电站图。（经密西根大学建筑及规划研究实验室Harold Borkin许可。）



a) b)

图1-7 取自多媒体教科书的一个交互算法。a) 实现该算法的计算机代码，b) 一个画草图的交互面板，当读者指到代码时，它即出现。指定一线段的终点时，代码将执行

1.2 计算机图形学的简单回顾

2~6 本书侧重于那些源于过去，至今仍在使用，并且将来大概还会继续使用的基本原理与技术。在本节中，我们简要地回顾计算机图形学的发展历史，说明当今系统的由来。关于该领域令人兴趣盎然的演化过程在文献 [PRIN71]、[MACH78]、[CHAS81]和[CACM84] 中已有较为完整的论述。编写硬件的发展史显然比软件的要容易一些，因为硬件的进展对该领域如何发展影响较大。因此，我们就先从硬件谈起。

尽管在计算时代的早期存在硬拷贝设备（如电传机和行式打印机）上粗略地画图。1950年麻省理工学院（MIT）开发的旋风式计算机（Whirlwind Computer）配置了由计算机驱动的CRT显示器作为输出，既供操作员使用，也供照相制作硬拷贝。现代意义上的交互式图形学起源于Ivan Sutherland开创性的关于Sketchpad绘图系统博士论文[SUTH63]。他引入了用于存储由标准组件简单复制构成的多层符号的数据结构，这是一种类似于使用塑料模板画电路符号的技术。他还开发了利用键盘和光笔实现选择、定位和绘图等交互技术，并提出了其他许多沿用至今的基本原理和技术。的确，许多Sketchpad中引入的特征在第7章中将要讨论的PHIGS图形软件包

中仍可见到。

与此同时，计算机、汽车和飞机制造商们已经清楚地意识到，计算机辅助设计（CAD）和制造（CAM）对于自动绘图和需要大量绘图的工作具有巨大的潜力。通用汽车公司用于设计汽车的DAC系统[JACK64]和Itek公司用于设计透镜的Digitek系统，是表现工程中常见的迭代设计周期中图形交互卓有成效的先驱。到了20世纪60年代中期，已经出现了一些研究项目和商业产品。

由于那时的计算机输入/输出（I/O）主要用穿孔卡批处理完成，非常希望在交互式人机通信上有所突破。交互式图形学作为“计算机的窗口”，成为大幅度缩短交互设计周期不可缺少的一部分。然而，结果不如想像的那样显著，交互式图形仍被划归于技术密集的组织所属的资源范畴。

因此，直到20世纪80年代早期，计算机图形学还是一个狭窄而特殊的领域，这主要是因为当时的硬件十分昂贵，而且缺乏易用和廉价的基于图形的应用程序。此后，带有内置式光栅图形显示器的个人计算机（例如施乐Star，以及后来投入批量生产、价格较低的苹果Macintosh和IBM PC及其兼容机）使得人机交互中使用**位图图形**广为流行。位图是一个在屏幕上由0和1表述的矩形点阵，每一个点称为一个**像素**（pixel）或**图像单元**（pel）。位图图形一经出台，便很快导致了廉价易用的基于图形应用程序的大量涌现。基于图形的用户界面使数百万新用户得以驾驭简单、便宜的应用程序，例如，电子表格、字处理和制图程序。

桌面概念现在已成为一个为人熟知的组织屏幕空间的方式。借助**窗口管理器**，用户可以产生、定位并调整矩形屏幕的面积，称为**窗口**。其作用像一个虚拟图形终端，每一个窗口运行一个应用程序。这使得用户只要用鼠标或其他设备指向希望的窗口，就可以在多个活动间互相切换。像在一张杂乱书桌上的纸张一样，窗口可以任意地重叠。图标显示也是桌面隐喻的一部分，它不仅可表示数据文件和应用程序，还可表示公共的办公对象，例如文件夹、邮箱、打印机和垃圾箱，它们在计算机上的操作与其实际对应物的使用完全类似（参见图1-1）。 7

通过“**点击**”直接操纵对象取代了早期操作系统和计算机应用中许多神秘的键入命令。因此，用户可以选择图标来激活对应的程序或对象，或者选择下拉式或弹出式屏幕菜单中的按钮来做出选择。今天，几乎所有的交互式应用程序，甚至连那些操作文本（如字处理）或数值数据（如电子表格）的程序，在用户界面和显示及操纵特定应用对象时都广泛地使用了图形。

1.2.1 输出技术

20世纪60年代中期开发的一直到20世纪80年代中期还在使用的一种显示设备被称为**向量显示器**、**笔划显示器**、**线条绘制器显示器**或**书法显示器**。向量（vector）这个词，在这里被用作线条（line）的同义词；笔划（stroke）是短的线条，而字符则由一系列这样的笔划组成。一个典型的向量系统由显示处理器、显示缓冲存储器和显示屏（CRT）组成，显示处理器与中央处理器（CPU）相连，作为其外围输入/输出（I/O）。

向量系统的本质是按照显示命令的任意顺序，将电子束从一个端点偏转到另一个端点，这种技术称为**随机扫描**（见图1-10b）。由于荧光物质输出的衰减要持续数十或数百微秒，为了避免闪烁，显示处理器应以每秒至少30次（30 Hz）的频率循环执行显示列表，**刷新**显示器。

20世纪70年代初，基于电视技术发展起来的低价位光栅图形设备对于计算机图形学领域成长的贡献远远超过此前所有其他的技术。**光栅显示器**（raster display）将显示**图元**（如线、文字、填充涂色或图案区域）以其像素形式存储在一个刷新缓冲器中，如图1-8所示。在某些光栅显示器中，有一个硬件显示控制器（如图1-8所示），它接收和解释输出命令序列，类似于向量显示器。在简单、常用的系统（例如个人计算机）中，显示控制器仅以图形库软件包的软件

组件形式出现，刷新缓冲器仅仅是CPU内存中的片段，它能够被图像显示子系统（常被称为**视频控制器**）读出，在屏幕上产生实际的图像。

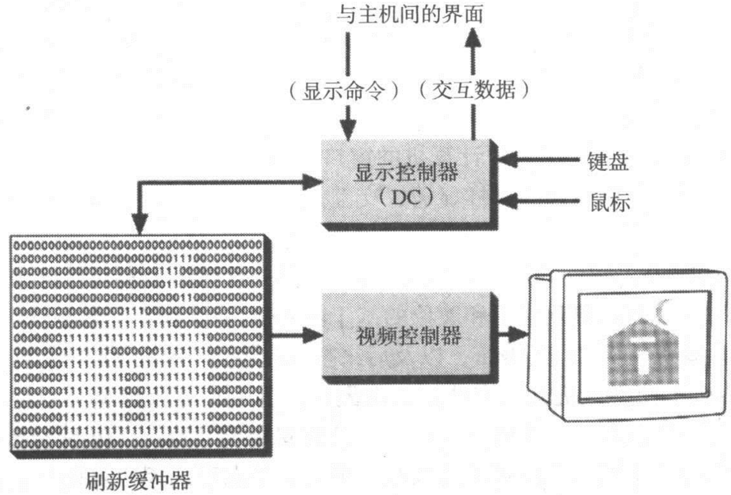


图1-8 光栅显示的体系结构

光栅显示器上的全部图像是由**光栅**（raster）形成的。光栅是一组互相平行的水平**扫描线**，每行有各自的像素；所以，光栅存储为一个代表了整个屏幕区域的像素矩阵。整幅图像由视频控制器按照从上到下然后再跳回顶部的顺序逐行扫描（如图1-9所示）。在每个像素上，电子束的亮度反映了像素的亮度；彩色系统中，三个电子束分别对应红、绿、蓝三原色，使之与每个像素值的三个颜色分量值相一致（见第4章和第11章）。图1-10a说明了随机扫描和光栅扫描在显示一所房子的简单二维线条时的差别。在图1-10b中，向量弧线段标上了箭头，表示电子束的随机偏转。虚线表示电子束的"**空**"偏转，因而偏转过程中，向量不画在屏幕上。图1-10c表示用矩形、多边形和圆弧绘制的未填充的房子，而图1-10d是一个填充的版本。注意光栅扫描图像（图1-10c）和（图1-10d）中直线和圆弧的锯齿形状，我们将简短地讨论这些人为的视觉缺陷。

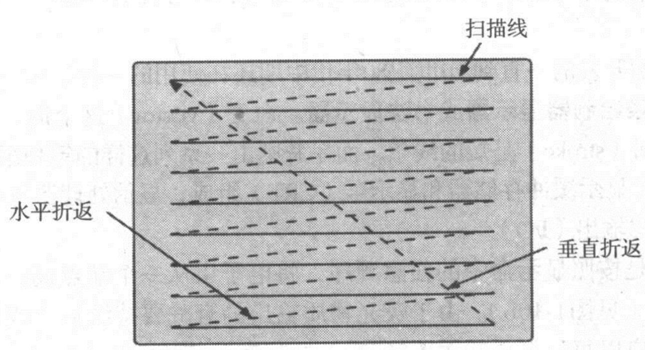


图1-9 光栅扫描

因为在光栅系统里，整幅图像，也就是1024行且每行1024个像素点，都必须直接保存到刷新缓冲器中，所以20世纪70年代初，便宜的固态随机访问存储器（RAM）被用于位图是一个突破，它导致了光栅图形成为占主导地位的硬件技术。**二值**（或称为**单色**）CRT使用黑白或黑绿两色绘图，有些等离子显示器则使用橙黑两种颜色。二值位图的每个像素占用单独一位，与一个分辨率

为1024 × 1024像素相对应的完整位图仅有2^{20}位，即大约128 000字节。低端彩色系统每个像素有8位，允许同时显示256种颜色；较高级的系统每个像素占用24位，可供选择的颜色有1600万种；现在，甚至个人计算机也使用了每个像素占32位，屏幕分辨率达1280 × 1024的刷新缓冲器。如第4章所述，32位中的24位被用来表示颜色，余下的8位用于控制。一个典型的每个像素占24位，分辨率为1280 × 1024的彩色系统需要3.75 MB的RAM，按现今的标准并不昂贵。严格地讲，**位图**这个术语只适用于每个像素一位的二值系统；对于每个像素多位的系统，我们使用更广义的术语**像素图**（pixmap，pixel map的缩写）。照一般的说法，像素图既可指刷新缓冲器的内容，也可指缓冲存储器的本身，所以当我们指的是实际缓冲存储器时，我们使用术语**帧缓存**。

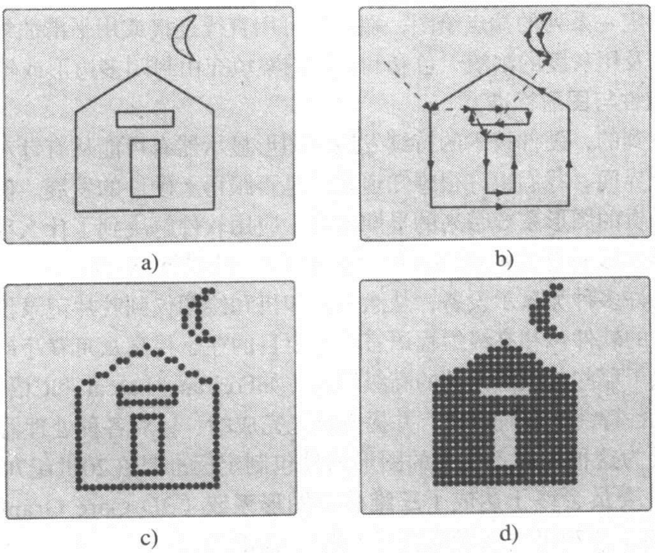


图1-10 随机扫描与光栅扫描。我们用填充了灰色的圆角矩形代表屏幕的白色背景，图像用黑色画在背景上 a) 理想的线图，b) 向量扫描，c) 有轮廓图元的光栅扫描，d) 有填充图元的光栅扫描

与向量图形相比，光栅图形的主要优点是造价低和具有对显示区域填充颜色或图案的能力。区域填充是一种极为有力的信息交流手段，对于显示三维真实感图形尤为重要。

与向量系统相比，光栅系统的主要缺点来自于像素表达的离散性质。首先，线和多边形这样的图元是用其端点（顶点）定义的，必须通过*扫描转换*为帧缓存中的像素单元。**扫描转换**是指这样一个概念，程序员以随机扫描方式输入图元中端点或顶点的坐标，这些图元信息再由系统简化为光栅扫描方式显示的像素。在个人计算机和低端工作站中，扫描转换通常由软件实现，微处理器CPU负责处理所有的图形。

光栅系统的另外一个缺点来自光栅本身的性质。向量系统能够从CRT屏上的任何一点与其他点之间画连续的、平滑的直线（甚至光滑的曲线）；但光栅系统在理论上只能用光栅网格上的像素近似地描绘平滑的直线、多边形和诸如圆与椭圆那样的曲线图元的边界。它引起了锯齿或阶梯状这个大家熟悉的问题，见图1-10c和图1-10d。这种视觉人工痕迹是信号处理理论中一种被称为**走样**的错误采样的表现。当用离散采样点来逼近一个亮度急剧变化的连续函数时，就会出现这种现象。现代计算机图形学要涉及灰阶系统和彩色系统的“*反走样*”技术。这些技术是使图元边缘邻近像素点上的亮度逐渐过渡，而不仅是将这些像素点的亮度值置为最大或零。对于这个重要主题的深入讨论，请参见第3章。

1.2.2 输入技术

过去这些年，输入技术也有了很大的改进。向量系统中笨拙、脆弱的光笔已被无处不在的鼠标（最早是由办公自动化的先驱Doug Engelbart于20世纪60年代中期开发出来的[ENGE68]）、数据输入板和贴在屏幕上的透明触敏膜所取代。甚至连那些不仅能够输入屏幕上的（x, y）坐标，而且还能输入三维甚至更高维数的输入值（自由度）的新式输入设备都变得很普遍（将在第8章中讨论）。声音通信也有令人激动的潜力，因为它允许不用手的输入、简单指令的自然输出以及反馈等。借助标准的输入设备，用户可以通过键入或画新的信息或指向屏幕上既存的信息，来指定
8~11 操作或图像成分。这些交互不需要编程知识，并且只需少量使用键盘：用户通过简单地选择菜单按钮或图标做出选择，通过检查选择项或在表格中键入少量字符来回答问题，在屏幕上粘贴预先定义的符号；通过指定一系列的端点绘图，端点间可用直线连接或用平滑曲线插值，通过在屏幕上移动光标涂色，以及用灰度的浓淡、色彩和各种图案填充由封闭多边形或轮廓线包围的区域。

1.2.3 软件的可移植性与图形标准

正如我们已经看到的，硬件技术的持续发展使图形显示器有可能从有针对性的特殊输出设备演化成为标准的人机界面。我们也许很想知道软件是否跟得上硬件的发展。例如，对于那些由过于复杂、麻烦而且昂贵的图形系统带来的早期困难，应用软件解决到了什么程度？我们已经从由制造商为其特殊的显示设备开发的低层次的设备相关软件包转向更高层次的设备无关的软件包。这些软件包能够驱动许多种类显示设备，从激光打印机和绘图仪到胶片记录仪和高性能实时显示器。使用与设备无关的软件包和高级编程语言的主要目的在于提高*应用程序的可移植性*。这种可移植性是以非常类似于那些与机型无关的高级语言（如Fortran、Pascal 和C语言）的方式提供的：把程序员从大量的设备特性中解脱出来，并提供预先完成的、针对各种处理器的语言特征。

普遍意识到需要为这种与设备无关的图形软件包制定标准是在20世纪70年代中期。1977年ACM SIGGRAPH⊖委员会终于达成了**三维核心图形系统**（3D Core Graphics System，简称Core）规范[GSPC77]，1979年对该规范进行了细化[GSPC79]。

Core规范起到了基本规范的作用。它不仅提供了许多方法，而且在ANSI（美国国家标准协会）和ISO（国际标准化组织）内部，也被用作官方（政府）标准项目的输入。第一个成为官方标准的图形规范是GKS（图形核心系统）[ANSI85]。它是一个经过精心制作，由Core简化而成的标准。与Core不同的是，它只局限于二维。1988年GKS-3D[INTE88]（一个GKS的三维扩展）成了官方标准；与此同时，一个更完善但也更复杂的图形系统，即PHIGS（程序员分层交
12 互图形系统[ANSI88]），也成为官方标准。GKS支持把逻辑上有关联的图元（如直线、多边形和字符串）与它们的属性聚集在一起，称为**段**（segment），但这些段不可以嵌套。正如其名称所示，PHIGS支持三维图元的嵌套式多层分组，称为**结构**（structure）。在PHIGS中，为了实现动态运动，所有的图元，包括子结构的引用，都要经过几何变换（缩放、旋转和位移）。PHIGS也支持结构的保留数据库，程序员可以有选择地编辑这些结构；任何时候只要数据库发生了变化，PHIGS就会自动更新屏幕。PHIGS已经被扩展到带有现代的、在光栅显示器上对物体进行伪真实感绘制⊜的若干特性。这一扩展称为PHIGS + [PHIG88]，它早于提交给ANSI/ISO

⊖ SIGGRAPH是Special Interest Group on Graphics（图形特别兴趣小组）的简称，它是计算机学会（Association for Computing Machinery, ACM）中的一个专业群体。ACM是计算机专业人员的两大专业团体之一，另一个团体为国际电气与电子工程师协会（IEEE）的计算机分会（IEEE Computer Society）。SIGGRAPH出版研究杂志，并主办一个年会发表和展出该领域的研究论文与设备。IEEE的计算机分会也出版图形学研究杂志。

⊜ *伪真实感绘制*仅模拟物体对光线反射的简单光学定律，而*真实感绘制*则需要更准确地近似物体对光线的反射和折射。这样的近似计算量较大，但生成图像的质量更接近于摄影图片（见彩图E）。

和ISO中的PHIGS PLUS [PHIG92]。由于许多特性和规范的复杂性，PHIGS的实现表现为大量的软件包。PHIGS和PHIGS PLUS在有硬件支持其变换、裁剪和绘制功能时，运行最佳。

除了由国家、国际或专业组织团体发布的官方标准外，还有非官方标准。这些所谓工业标准或事实上的标准是由独立的公司或公司和大学的联盟开发、升级和得到许可的。熟悉的工业图形标准包括Adobe公司的 PostScript，Silicon Graphics公司的OpenGL，Ithaca Software 公司的HOOPS，由MIT 牵头的X联盟的 X Window System及其客户方/服务器协议3D图形扩充 PEX（见第7章）。由于工业标准能更快地更新，因此它可能比官方标准更流行，在商业上也更重要，特别是关联到一个公司的关键商业产品及其随后相当数量资源的那些标准。

本书将用一些篇幅讨论图形软件的标准。我们首先在第2章中研究简单光栅图形软件包（Simple Raster Graphics Package，SRGP），它是一种仿照苹果计算机的QuickDraw整数光栅图形软件包[ROSE85]和麻省理工学院的X Window System（X 视窗系统）[SCHE88]的功能作为输出、仿照GKS和PHIGS的功能作为输入的软件。在看过这种低层光栅图形软件包的应用之后，我们再来研究这些软件包中用来在帧缓存中产生图元图像的扫描转换和裁剪算法。在建立起二维和三维几何变换以及三维平行投影和透视投影的数学基础之后，我们再研究一个功能更加强大的软件包SPHIGS（Simple PHIGS）。SPHIGS是PHIGS的一个子集，它以定义在一个独立于任何显示技术的浮点、抽象和三维世界坐标系统中的图元为操作对象，并且支持一些简单的PHIGS PLUS功能。我们的讨论针对PHIGS和PHIGS PLUS，因为我们相信它们对交互式三维图形的影响比GKS-3D要大得多，特别是在更多地使用硬件支持实时变换和绘制伪真实感图像的情况下更是如此。 13

1.3 交互式图形学的优点

图形给人类与计算机的交流提供了一种最为自然的方式，因为人类高度发达的二维和三维模式识别能力使我们可以快速、高效地理解和处理图像数据。在当今的许多设计、实现和构筑过程中，图像信息实际上是不可缺少的。科学计算可视化在20世纪80年代后期成为一个重要的领域，科学家和工程师们知道，如果不对数据进行归纳，并使用各种图形表现方法，他们就无法理解由超级计算机运算产生的大量数据。

交互式的计算机图形是自摄影和电视发明以来最重要的图形方法。它有许多额外的优点。利用计算机，我们不仅可以制作有形的“真实世界”物体的图像，还可以制作抽象的合成物体的图像，例如，用图形表示调查结果数据。此外，计算机不仅产生静态图像，还可以产生动态图像。虽然静态图像是交流信息的一种好方法，但动态图像往往效果更好。可以这样讲，一幅动态图像抵得上上万幅静态图像。这句话特别适用于时变现象，无论是真实的过程（如超音速飞行中飞机机翼的变形或人的脸部从童年到老年的演变），还是抽象的过程（如美国核能使用量或城乡间人口迁徙的增长趋势）。所以，许多交互式的图形技术都包含了用户控制的运动动力学和更新动力学的硬件和软件。

运动动力学（motion dynamics）的方法是，使物体相对于静止的观察者移动或旋转；也可保持物体不动，而使观察者围绕物体移动、摇动镜头以选择视框或者拉近或推远景物以选择细节的多寡，这就像快速移动摄像机时，通过瞄准镜所看到的情景一样。在很多情况下，物体和观察者都在运动。一个典型的例子是飞行模拟器（彩图4a和彩图4b），它由一台支撑着一个模拟飞机座舱的机械平台和若干窗式显示屏组成。计算机控制平台的运动及运动的规则，并模拟飞行员航行中看到的静止和运动的周围世界。游乐场也提供一些在陆地的或宇宙的景观中驰骋的“运动模拟器”。视频娱乐厅提供了基于图形的敏捷性游戏和赛车驾驶模拟器（彩图5）（一种利

用交互运动动力学的视频游戏）（见彩图5）。

更新动力学（update dynamics）是指被观察物体的形状、颜色或其他性质实际发生了变化。例如，一个系统可以显示一架飞机的结构在飞行中的变形，或一个核反应堆流程中的状态因操作者在许多控制机构的图形界面上的操作而发生的变化。变化过程越平滑，其结果就越真实、越有意义。

因此，交互式计算机图形允许大量的、高带宽人机交互。这会明显地增强我们理解数据、
14 把握趋势和观看真实或假想对象的能力。

1.4 交互式图形学的概念框架

图1-11所示的高级概念框架可被用来描述几乎任何一种交互式图形系统。在硬件层次上（图中未明确画出），计算机从交互设备接收输入，并向显示设备输出图像。软件有三个部分：第一部分是**应用程序**，它创建（图元），并将之存入应用模型或由应用模型取回（图元）。**应用模型**是软件的第二部分，代表了将在屏幕上显示的数据和物体。应用程序也处理用户的输入。它通过向图形系统发送一系列图像输出命令产生视图，这些命令既包括了将被观看对象的详细几何描述，也包括了描述这些对象将如何显现的属性特征。**图形系统**是软件的第三部分，负责从对象的细节描述产生实际的图像，并将用户的输入传递给应用程序，以供处理。

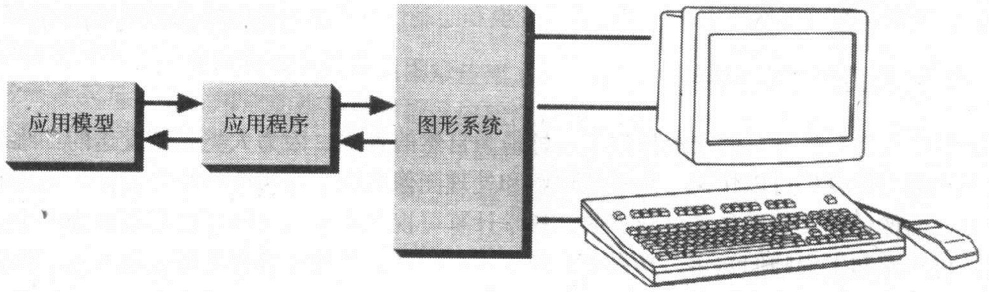


图1-11 交互式图形的概念框架

因此，图形系统是应用程序和显示硬件之间的中介，起着将应用模型中的物体转化为模型视图的**输出变换**作用。与此相对称，它还起着把用户的活动转变为应用程序的输入的**输入变换**作用，这些输入将使应用软件改变其模型或图像。一个交互式图形系统应用程序设计者的基本任务是，规定生成哪些类的数据项或物体，并将它们表示为图像，以及说明用户和应用程序之间如何交互，以创建和修改模型及其视觉表达。大多数程序员只涉及创建和编辑模型和处理用
15 户交互，并不实际创建视图，因为那是由图形系统处理的。

1.4.1 应用建模

应用模型捕捉所有的数据、物体和它们之间的相互关系，这些内容与显示和应用程序的交互部分相关，而且与非图形后处理模块有关联。后处理模块可以是分析一个电路或一个机翼中应力的瞬态行为、人口模型和气象系统的仿真和建筑物的价格计算。在标有画图字眼的一类应用程序中，如MacPaint和PCPaint，程序的目的是通过让用户设置或修改每一个像素来产生图像。这里，不需要一个明确的应用模型，图像本身既是手段又是结果，而且被显示的位图或像素图起着应用模型的作用。

然而，还有一种更典型的等价应用模型，它通过数据和独立于特殊显示设备的过程描述的

组合表达应用对象。例如，过程描述被用来定义分形，如14.11节所述。数据模型可以像数据点的数组那样简单，也可以像链表、网状数据结构或存储了许多关系的关联数据库那样复杂。我们常说将**应用模型**存入**应用数据库**，这两个术语在这里可以互换使用。模型通常存储图元的描述（二维或三维的点、线和多边形及三维的多面体和自由曲面），它们定义了物体中各组成部分的形状，物体的线型、颜色或表面纹理等**属性**，以及说明物体各部分如何组装到一起的**连接**关系与位置数据。

应用模型中的几何数据经常伴随着非几何的文本或数值特征信息，这些信息对于后处理程序或交互式的用户非常有用。例如，在CAD应用中，这种数据包括制造数据，价格与供应商数据，热、机械、电气或电子特性，以及机械或电气公差。

1.4.2 模型的显示

应用程序创建应用模型，或是在前面的计算阶段预先完成，例如超级计算机上的工程或科学仿真，或是作为交互会话的一部分，在显示设备上由用户逐步引导，选择元件和几何及非几何特性数据。用户可以随时要求应用程序显示已生成模型的视图（view）。（这里有意使用“视图”这个词，既指建模对象的某些几何特性的视觉形象，也指技术数据库中模型的某些子集的某些特性的二维表达。）

模型因应用而异，其产生与具体的显示系统无关。因此，应用程序必须把模型中要被观察的部分从内部的几何表示（无论是在模型中显式存储，还是随用随导出）转换为图形系统用来生成图像的过程调用或命令。这个转换过程分为两步。首先，应用程序遍历存储模型的应用数据库， 16
用一些选择或查询准则抽出要观察的部分。然后，抽出的几何形状被转换为可以向图形系统发送的格式。选择的准则可以是自然界中的几何形状（例如，要被观察的模型的部分通过图形发生移动，等效于照相机的镜头摇动和景物拉伸操作），或者也可以类似于传统数据库的查询准则。

在数据库遍历期间取出的数据必须是几何形状的数据、或者必须已转换为几何的数据。这些数据可用图形系统直接显示的基本图元以及控制这些图元外观属性的形式记入图形系统。显示图元一般与几何模型中存储的图元相匹配。这些图元包括：二维的线、矩形、多边形、圆、椭圆和文本，以及三维的多边形、多面体和文本。

图形系统一般包括一组与各种图元、属性和其他要素相对应的输出子程序。它们被收集在**图形子程序库**或**软件包**中，可被C、Pascal和LISP等高级语言调用。应用程序向这些子程序指定几何图元和属性，然后这些子程序驱动特定的显示设备，并使之显示图像。就像常规的I/O系统创建逻辑I/O单元使应用程序员不必面对大量的硬件和设备驱动细节一样，图形系统也要创建一个**逻辑显示设备**。这种显示设备的抽象既适合于图像的输出，也适合于通过逻辑输入设备进行的交互。例如，鼠标、数据输入板、触摸板、2D游戏杆或轨迹球都可以被视为返回屏幕坐标(x, y)的位置的**定位器**逻辑输入设备。应用程序可以要求图形系统对输入设备进行**采样**，或者在一个特定的点等待，直到发生了用户操作处于等待状态设备的**事件**。

1.4.3 交互处理

典型的应用程序交互处理方法为**事件驱动循环**。它可以被简单地看作是一台有限状态机，具有一个中央等待状态，用户输入事件使之转移到其他状态。处理一条命令可能需要一连串具有相同格式事件的嵌套循环，它们拥有各自的状态和输入转移。一个应用程序也可能通过随时查询它们的数值对输入设备（例如定位器）进行采样；然后程序使用返回值作为处理过程的输入。处理过程也改变应用程序、图像或数据库的状态。事件驱动循环可以用下面的程序伪代码来表示：

```
产生初始显示，由应用模型适当地导出
do
    使选择命令有效
    /*程序停顿于“等待状态”，直至用户干预*/
17  等待用户选择
    switch(选择){
        执行选择以完成命令或执行已完成的命令，
        按要求更新模型和屏幕
    }
}
while (!退出)  /*用户没有选择“退出”操作*/
```

让我们更详细地查看应用程序对输入的反应。应用程序通常以一种或两种模式响应用户的交互。首先，用户的动作可能仅要求屏幕更新，例如，通过点亮选中的对象或通过生成新的选择菜单。那么，应用只需要更新它的内部状态，并调用图形软件包去更新屏幕；而不必更新数据库。然而，如果用户要求改变模块，例如增加或者删除组件，应用就必须更新模型，然后调用图形软件包从模型去更新屏幕。或者遍历整个模型重绘图像，或者应用更复杂的增量更新算法，有选择地更新屏幕。如果没有模型的变化，屏幕上显示的对象就不会发生明显的变化，理解这一点非常重要。屏幕的确是计算机的窗口。在其中，用户通常操作的不是图像，而是字面和象征意义上隐藏在图像后面的模型。只有在绘画和图像增强应用中，图像和模型才是等同的。所以，应用程序的工作是解释用户的输入。图形系统没有责任在最初或在与用户的交互中，建立和修改模型；它的惟一的工作仅仅是从几何描述中创建图像，并读入用户的输入数据。

事件循环模型，尽管对现在计算机图形应用来说很基本，仍有一定的局限性。其人机对话是一种顺序乒乓模式，用户动作与计算机反应轮流进行。将来，我们可能期望看到更多的并行交谈，利用多种通信信道（例如图形和语音）同时进行输入和输出。无须顾及编程语言的结构，进行这种自由形式交谈的形式目前尚未确立，这里我们将不做更多的讨论。

小结

图形界面已经取代了文本界面，成为人机交互的标准手段。在商业、科学、工程和教育等许多领域中，图形也已成为一项交流思想、数据和趋势的关键技术。利用图形，我们可以创建人工现实，进行基于计算机的“探险”，用自然和直观的方法检验物体和现象，这种方法则利
18 用了我们在视觉模式识别中高度发达的技巧。

直到20世纪80年代后期，大量的计算机图形应用还只是处理二维对象，三维应用相对稀少，这既是因为三维软件本来就比二维软件复杂得多，也是因为绘制接近真实的图像需要大量的计算能力。因此，直到最近，用户与三维模型和伪真实感图像间的实时交互，只是在一些带有专用图形硬件并且非常昂贵的高性能工作站上才获得使用。担负发展廉价微处理器和存储器责任的VLSI半导体技术的引人注目的进步，导致了20世纪80年代早期基于二维位图图形的个人计算机的出现。同样的技术在不到10年之后，已使得仅用几个芯片制成的图形子系统成为可能，该子系统能够绘制含有数千个多边形的复杂物体的彩色光照图像，实现了实时三维动画。图形子系统可以作为三维加速器附加在工作站甚至是使用普通商业芯片的个人计算机上。很明显，三维应用的爆炸性成长将会与当前的二维应用的成长形成并存局面。而且，在本书1982年版中还认为是非同寻常的照片真实感绘制，现在已经成为当前技术的一部分，可在图形软件和日益增多的图形硬件中使用。

实现有效的图形交流的首要任务，无论是二维还是三维，在于建立我们想要产生图像的对象的模型。图形系统在应用模型和输出设备之间，起着中介的作用。应用程序负责根据用户的交互作用创建和更新模型。在生成对象的视图和向应用传递用户事件的过程中，图形系统完成容易理解的常规工作。日益增多的有关各种基于物理建模的文献表明图形已远远超出了绘制和交互处理的范畴。图像和动画不再仅仅是科学和工程中的图示，它们已成为了科学和工程的内容中的一部分，并且影响着科学家和工程师们如何管理他们的日常工作。

习题

1.1 列出在你的“知识工作”中经常使用到的交互式图形程序，例如书写、计算、绘图、编程和调试等。其中哪些程序也可以在文字终端上实现？如果没有图形能力，哪些程序将几乎无法工作？请解释您的回答。

1.2 词组“视感”已被广泛用于图形程序的用户界面中。详细列出你喜爱的字处理或窗口管理程序的图形界面的主要组件的外观，如图标、窗口、滚动条和菜单等。列出这些窗口小部件所需图形能力的种类。你看到有什么机会将颜色和三维绘制应用于这些外观吗？例如，如何使杂乱的办公室在空间上更好地隐喻，以易于组织和存取信息，而不再是一个“凌乱的桌面”？ 19

1.3 与习题1.2类似，你看到有什么机会用动态图标增强或取代现有桌面隐喻中的静态图标？

1.4 利用图1-11所示的概念模型作为引导，将你喜爱的图形应用拆解为主要的模块。有多少模块实际是处理图形的？又有多少是处理数据结构生成与维护的？有多少是处理计算和仿真的？

1.5 “仿真”与“动画”两个术语在计算机图形中经常一起使用，有时甚至可以互换。在显示某些物理的或抽象的系统的行为（或结构）随时间变化时，这是自然的。请构造一些能够体现可视化优点的示例系统，确定仿真的形式和运行方法。给出一个仿真和动画相区分的例子。

1.6 作为习题1.5 的变形，请为科学、数学或工程领域的一个重要主题创建一个图形“探索”的高级设计。讨论交互顺序如何工作以及用户试验中应具备哪些工具。

1.7 不看第3章，写出一个直接的算法用于扫描转换一条位于第一象限的直线，直线的斜率小于等于45°。

1.8 走样是一个严重的问题，它产生难看的甚至令人误解的视觉失真。讨论在哪些情况下，会或不会发生走样。讨论使锯齿效应最小的各种方法，并解释这些补救措施的“代价”可能是多大。 20

第2章 简单光栅图形软件包（SRGP）的编程

在第1章，我们已经介绍了采用不同的硬件技术生成屏幕上图像的向量显示器和光栅显示器。因为光栅显示器有许多适合现代应用的特点，所以它成了目前的主流硬件技术。首先，光栅显示器既可以用同一种颜色均匀填充区域，也可以用具有两种或两种以上颜色的重复图案填充区域；而向量显示器最多只能用空间中平行的向量序列来模拟区域填充的效果。其次，光栅显示器存储的图像易于操作：既可对单独的像素进行读或写的操作，也可拷贝或移动图像中的任意部分。

我们首先要讨论的图形软件包SRGP（Simple Raster Graphics Package），是与设备无关的软件包，它利用了光栅显示技术。SRGP图元（直线、矩形、圆、椭圆和文本串）的组成类似于流行的Macintosh QuickDraw光栅软件包和X Window System中的Xlib 图形软件包。另一方面，它的交互处理功能只是用于显示三维图元的图形软件包SPHIGS（将在第7章介绍该软件包）的一部分。SPHIGS （Simple PHIGS）实际上是可同时用于向量和光栅硬件标准的图形软件包PHIGS（Programmer's Hierarchical Interactive Graphics System）的简化版本。尽管SRGP和SPHIGS是专为本书写的，但它们也是主流图形软件包的精髓，这里所学到的技术可以立即应用于其他商业软件包。在本书中，我们将介绍这两个软件包，有关它们的完整信息，请参考随软件包发行的用户手册。

我们从考察应用软件在屏幕上绘制图形所进行的操作入手来讨论SRGP：图元的规格以及影响图像的属性。（由于图形打印机的信息显示原理基本类似于光栅显示器，所以此处我们仅 21
考虑光栅显示器，而图形打印机将在第4章中介绍。）然后，我们将学习如何使用SRGP的输入功能使应用程序具有较好的交互性。接着介绍仅适合光栅显示器的像素操纵的实用工具。最后，我们将讨论整数光栅图形软件包（如SRGP）的局限性。

虽然在讨论SRGP时，我们假设它控制整个屏幕，但是我们是在窗口环境中（见第10章）设计软件包，因此可以将窗口的内部看作是虚拟屏幕，进而可以控制窗口的内部。因而应用程序员不必关心在窗口管理器控制下的运行细节。

2.1 用SRGP画图

2.1.1 图形图元的规格

用SRGP这样的整数光栅图形软件包绘图类似于在方格纸上绘图。在常规的显示器中坐标网格的变化范围是每英寸包含80~120个点，在高分辨率显示器中的变化范围是80~300个点。显示器的分辨率越高，图形显示效果越好。图2-1给出了在SRGP的整数笛卡儿坐标系统下的显示屏幕。SRGP的像素位于网格线的交点。

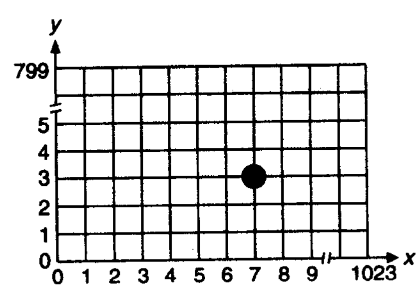


图 2-1 一个宽1024像素、高800像素的显示屏幕的笛卡儿坐标系统，图中标出了像素（7,3）

原点（0,0）位于屏幕的左下角；x轴正向向右增加，y轴正向向上增加。右上角像素点坐标为（width−1, height−1），其中，width（宽）和height（高）是与设备相关的屏幕尺寸。

在方格纸上，我们可以在任意两点之间画一条连续的线；然而，在光栅显示器上，我们只能在网格点间画线，而且只能用在直线上的点或最靠近直线的点来近似代表这一直线。同样，像多边形区域或圆这样的填充图形由其内部的和边界上的网格点像素生成。因为指定直线或封闭图形的每一像素非常麻烦，所以图形软件包只要求程序员指定图元，如给定了顶点的直线和多边形，然后由软件包利用扫描转换算法（将在第3章讨论）实现具体的细节。

SRGP支持一个基本的图元集合：直线、多边形、圆、椭圆和文本。[⊖] 为了指定一图元，应用软件需要将定义图元形状的坐标传给SRGP中恰当的图元生成程序。当指定的点位于屏幕的矩形区域外时也是合法的，但只有位于屏幕内的图元部分是可见的。

我们使用的ANSI C采用下面的排版约定。C的关键字、内部类型和用户定义类型用黑体。程序体中使用的变量用斜体表示。符号常数用大写字母。注释写在定界符/*...*/内。伪代码用斜体（中文注释用楷体）。对于明显的情况，为简化起见将省略常数和变量的声明。作为unsigned char类型的布尔变量，其值 TRUE和 FALSE分别地用1和0来定义。新的类型如point一旦定义了，那么在后面的代码段使用时，将不用再声明。

1. 直线和多边形

以下SRGP程序画一条从点（$x1, y1$）到（$x2, y2$）的线段：

```
void SRGP_lineCoord ( int x1, int y1, int x2, int y2 );
```

因此，要画一条从（0,0）到（100,300）的线段，我们只需简单地调用

```
SRGP_lineCoord (0, 0, 100, 300);
```

用线段顶点来代替单个的x，y坐标是很自然的想法，因此，SRGP还提供了如下的线段绘制函数：

```
void SRGP_line ( point pt1, point pt2 );
```

这里，“point”是已定义的结构类型，该结构包含两个整数，分别用于表示点的x，y坐标值。

```
typedef struct {
    int x, y;
} point;
```

22~23 连接一系列点的一组线段称为**折线**。尽管通过反复调用线段生成程序可以生成折线，SRGP还是对折线进行了特殊处理。SRGP中有两个折线生成程序，类似于线段生成程序的参数形式，该程序将数组作为参数。

```
void SRGP_polyLineCoord ( int vertexCount, vertexCoordinateList xArray,
        vertexCoordinateList yArray);
void SRGP_polyLine ( int vertexCount, vertexList vertices );
```

其中*vertexCoordinatelist*和*vertexList*是由SRGR包定义的类型，分别是整数数组和点数组。

折线生成程序中的第一个参数通知SRGP输入的顶点数。在第一个调用程序中，第二和第三个参数分别是输入点的x坐标数组和y坐标数组。折线从顶点（xArray[0], yArray[0]）到（xArray[1], yArray[1]），到（xArray[2], yArray[2]）等。这种形式非常方便，例如，当绘制的数据是位于轴上的标准集时，其中xArray是预先决定的独立变量值集，yArray是用户输入或计算

⊖ 专门用来画单独一个像素或一组像素的过程在SRGP参考手册里有所描述。

得出的数据集。

举个例子，让我们来绘制一个经济分析程序的输出，该程序计算每个月的交易数据，并将其存储在*balanceOfTrade* 数组中，该数组包含12个整数数据。我们从点（200, 200）开始绘制，为了能够看出连续两点之间的区别，在x轴上以10个像素点为间距绘制每一点。因而，我们生成一个整数数组*months*来表示12个月，并将所需的x值200, 210, …, 310存入数组中。类似地，y方向上的12个值均增加200。可以用以下代码绘制出图2-2中的图形：

```
/* 画坐标轴 */
SRGP_lineCoord (175, 200, 320, 200);
SRGP_lineCoord (200, 140, 200, 280);

/* 画数据 */
SRGP_polyLineCoord (12, months, balanceOfTrade);
```

我们也可以将x和y坐标值对作为点的定义，将点数组传给SRGP，通过调用第二种形式的折线绘制程序来绘制通过这些点的图形。通过调用如下的折线生成程序：

```
SRGP_polyLine (7, bowtieArray);
```

我们生成了图2-3中蝴蝶结。图2-3中的表格给出了*bowtieArray*的定义。

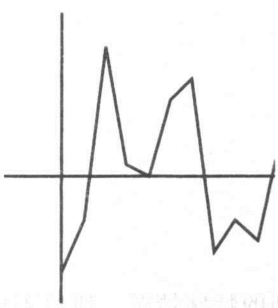
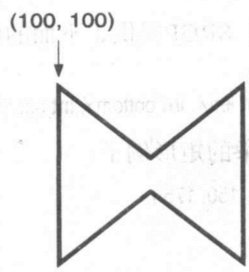


图2-2 绘制数据点列



	x	y
0	100	100
1	100	60
2	120	76
3	140	60
4	140	100
5	120	84
6	100	100

bowtieArray

图2-3 绘制折线

2. 标记和多点标记

在图形的数据点处设置标记（例如，点、星号或圆）通常是十分有用的。因而，SRGP提供了与直线和折线生成程序相配套的标记函数。以下程序在点（x, y）处生成标记符号：

```
void SRGP_markerCoord ( int x, int y );
void SRGP_marker ( point pt );
```

可以用2.1.2节介绍的方法改变标记的风格和大小。调用以下程序中的任意一个都可以在一组点上生成相同的标记。

```
void SRGP_polyMarkerCoord( int vertexCount, vertexCoordinateList xArray,
    vertexCoordinateList yArray );
void SRGP_polyMarker ( int vertexCount, vertexList vertices );
```

所以，调用以下过程可以在图2-2中的图形上生成标记而得到图2-4：

```
SRGP_polyMarkerCoord (12, months, balanceOfTrade);
```

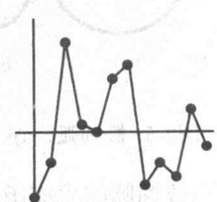


图2-4 用标记绘制数据点列

3. 多边形和矩形

为了绘制一多边形，我们可以定义一条首尾顶点相同的折线（和图2-3中一样），或者调用

以下SRGP函数：

```
void SRGP_polygon ( int vertexCount, vertexList vertices );
```

这一调用将自动在首顶点和尾顶点之间画一条线段。现在只需六个点，我们就可以画出图2-3中的蝴蝶结。

```
SRGP_polygon (6, bowtieArray);
```

任何一个矩形（rectangle）可以定义为有四个顶点的多边形，而只需两点（左下角和右上角），就可以利用SRGP的“矩形”图元画出一个正矩形（边平行于屏幕边界）。

24～25

```
void SRGP_rectangleCoord ( int leftX, int bottomY, int rightX, int topY );
void SRGP_rectanglePt ( point bottomLeft, point topRight );
void SRGP_rectangle (rectangle rect );
```

其中的“rectangle”记录存储了待绘制矩形的左下角和右上角坐标：

```
typedef struct {
    point bottomLeft, topRight;
} rectangle;
```

以下调用可以画出一个宽101个像素，高151个像素的正矩形：

```
SRGP_rectangleCoord (50, 25, 150, 175);
```

为了由坐标数据创建矩形和点，SRGP提供了下面的函数：

```
point SRGP_defPoint ( int x, int y );
rectangle SRGP_defRectangle ( int leftX, int bottomY, int rightX, int topY );
```

调用以下函数可以生成我们所举的矩形例子：

```
rect = SRGP_defRectangle ( 50, 25, 150, 175 );
SRGP_rectangle (rect);
```

4. 圆和椭圆

图2-5给出了由SRGP绘制的圆弧和椭圆弧。因为圆是椭圆的特殊情况，所以不论是圆或椭圆，也不论它是否封闭，我们均用椭圆弧代表。SRGP只能画主轴平行于坐标轴的标准椭圆。

尽管在数学上有许多方法定义椭圆弧，但是对于程序员来说，用椭圆的外接正矩形定义它是比较方便的（如图2-6所示）；这一正矩形称为包围盒或范围。

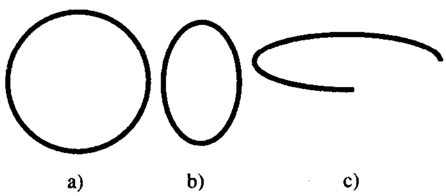


图2-5 椭圆弧。a)圆，b)椭圆，c)椭圆弧

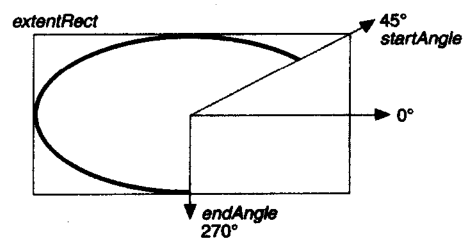


图2-6 界定椭圆弧

一般椭圆的生成函数如下：

```
void SRGP_ellipseArc ( rectangle extentRect, float startAngle, float endAngle );
```

范围的宽度和高度决定了椭圆的形状。弧的起始角和终止角决定曲线是否封闭。为方便起见，使用逆时针的矩形角来度量每一角度，x轴的正向对应0°，y轴的正向对应90°，从原点到右

上角的对角线对应45°。显然仅当范围是正方形时，矩形角等价于圆弧角。矩形角与圆弧角之间的一般关系是：

$$矩形角 = \arctan\left(\tan(圆弧角) \cdot \frac{宽度}{高度}\right) + adjust$$

其中，角度采用弧度表示，并且

$$adjust = 0, 0 \leqslant 圆弧角 < \frac{\pi}{2}$$

$$adjust = \pi, \frac{\pi}{2} \leqslant 圆弧角 < \frac{3\pi}{2}$$

$$adjust = 2\pi, \frac{3\pi}{2} \leqslant 圆弧角 < 2\pi$$

2.1.2 属性

1. 线型和线宽

图元的外观由它的**属性**⊖控制，在SRGP中，直线、折线、多边形、矩形和椭圆弧的属性包括线型、线宽、颜色和画笔类型。

属性是模态上设定的，即属性是一种全局状态变量，它将一直保持原值直至被显式地改变。图元定义后，将使用当时有效的属性值来绘制；因此改变属性值决不会改变以前创建的图元——它仅影响属性值改变后才定义的图元。采用模态属性十分方便，因为程序员不必为每一个图元都设定长长的参数列表（通常一个生产系统中可能有几十种不同的属性）。 26~27

调用以下函数可以设置线型和线宽。

```
void SRGP_setLineStyle ( lineStyle CONTINUOUS / DASHED / DOTTED / ... ); ⊜
void SRGP_setLineWidth ( int widthValue );
```

线的宽度由屏幕单元度量，即用像素度量。每一个属性都有一个默认值：线型的默认值是CONTINUOUS（连续线），线宽为1。图2-7显示了具有不同线型和线宽的直线。程序2-1给出了生成图2-7中图形的代码。

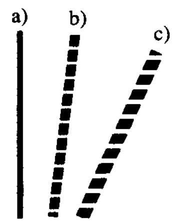


图2-7 不同线型和线宽的直线

我们可以认为线型是一个位掩码，使SRGP在对图元做扫描转换时，能够选择性地写像素。掩码中的零表示不写该像素，而保留存储在缓存中像素的原值。可以认为该像素是透明的，所以能看到底下的内容。因而CONTINUOUS对应的串位值均为1，DASHED对应的串为1111001111001111…，虚线长度为透明部分的两倍。

每一属性都有一个默认值；例如，线型的默认值是CONTINUOUS，线宽的默认值是1，等等。在前面的例子中，我们没有为绘制的第一条线段设置线型，因而我们使用的是线型的默认值。然而，在实际的应用中，假定当前的属性值是不安全的，在后面的过程中，我们将明确地设定每一属性，使程序标准化，便于调试和维护。在2.1.4节中，我们将看到，对于程序员来说，为每一个函数显式地保存和恢复属性值是比较安全的。

⊖ 这里对于SRGP属性，特别是不同属性的互相作用的描述经常缺少足够的细节。之所以缺少细节，是因为一个属性确切的效果是它的实现的一个函数，并且由于性能的原因，在不同的系统中使用了不同的实现；如果要进一步了解这方面的细节，请查阅专门的工具手册。

⊜ 在这里和以后的行文中，我们使用一个简化的符号。在SRGP里，这些符号常量实际上是枚举数据类型“lineStyle”的一个值。

程序2-1 生成图2-7中图形的代码

```
SRGP_setLineWidth (5);
SRGP_lineCoord (55, 5, 55, 295);          /* 直线a */
SRGP_setLineStyle(DASHED);
SRGP_setLineWidth(10);
SRGP_lineCoord(105, 5, 155, 295);         /* 直线b */
SRGP_setLineWidth(15);
SRGP_setLineStyle(DOTTED);
SRGP_lineCoord(155, 5, 285, 255);         /* 直线c */
```

可为标记图元设置的属性有

```
void SRGP_setMarkerSize ( int sizeValue );
void SRGP_setMarkerStyle ( markerStyle MARKER_CIRCLE
    / MARKER_SQUARE / ... );
```

标记的大小规定了包含该标记的正方形边的像素长度。参考手册中给出了全部标记类型的集合。
28 图2-4中所示的圆类型是标记的默认值。

2. *颜色*

目前所给出的属性只影响SRGP的一部分图元，但是整型颜色属性将影响全部图元。十分明显，颜色属性的意义很大程度上依赖于支持它的硬件设备；任一系统上的两个颜色值就是0和1。在二值显示系统中，这些颜色的外观很容易预测——对于黑白设备，颜色1代表像素是黑色的，颜色0代表像素是白色的。对于绿黑设备，颜色1代表像素是绿色的，颜色0代表像素是黑色的，等等。

整型颜色属性并不直接指定颜色值；更确切地说，它是SRGP**颜色表**的索引，该表中的每一项定义一个程序员不需要知道的颜色或灰度值。在颜色表中有2^d个条目，其中d是帧缓存的深度（每一像素的存储位数）。在二值设备上，颜色表固化在硬件上；然而，在多色系统中，SRGP允许应用程序修改颜色表。第4章将讨论颜色表提供的一些间接应用。

应用程序可以有两种方法确定颜色。如果该应用程序强调与硬件无关，则应当直接使用0和1确定颜色值，这样应用程序可同时在二值和彩色显示设备上运行。如果应用是颜色支持的，或者是为特定的显示设备编写的，那么它可以使用由SRGP支持的与实现相关的*颜色名*。这些名字都是符号常量，它们指示在特定设备的默认颜色表中标准颜色的位置。例如，黑白设备提供两个颜色名COLOR_BLACK（1）和COLOR_WHITE（0）；在本章的示例代码段中，我们将使用这两个值。注意对于修改颜色表的应用程序来说，颜色名字是没有用的。

可以通过调用以下函数选定颜色：

```
void SRGP_setColor ( int colorIndex );
```

2.1.3 填充图元及其属性

包含区域的图元（所谓的*区域定义图元*）可以用两种方法绘制：*描绘轮廓*或者*区域填充*。前一节给出的程序生成的是前一种类型的图形：轮廓封闭，内部没有填充。SRGP区域定义图元的填充版本绘制不含轮廓线的区域内部像素。图2-8给出了SRGP的填充图元的种类，包括填充椭圆弧，即薄片（pie slice）（图2-8b）。

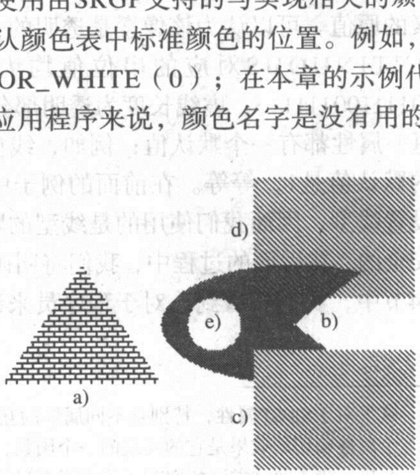


图2-8 填充图元。a)~c) 不透明的位图模式，d) 透明的位图模式，e) 实区域

注意到SRGP并不画出区域外轮廓，如一个像素宽的实线边界，所以需要绘制轮廓的应用程序必须再显式地画出它。是应该画出区域定义图元边界上的像素，还是仅画出严格位于内部的像素是一个敏感的问题。这一问题将在3.4节详细讨论。

为了生成填充多边形，我们使用函数SRGP_fillPolygon或者SRGP_fillPolygonCoord，它们的参数与非填充版本相同。用同样的方法可以定义其他区域填充图元，这时要在它们原有的函数名前添加前缀“fill”。因为多边形可能是凹的或者甚至自相交，所以我们需要一个规则来规定什么区域是填充内部并因此应该被填充，什么区域是填充外部。SRGP多边形遵循奇偶规则。为了决定一个区域位于一个给定多边形的内部还是外部，在这个区域内任意选择一个点作为测试点；接下来，从这个点向任意方向画一条射线，这条射线不经过任何顶点，如果这条射线与这个多边形的轮廓相交次数为奇数，这个区域被认为在其内部（见图2-9）。

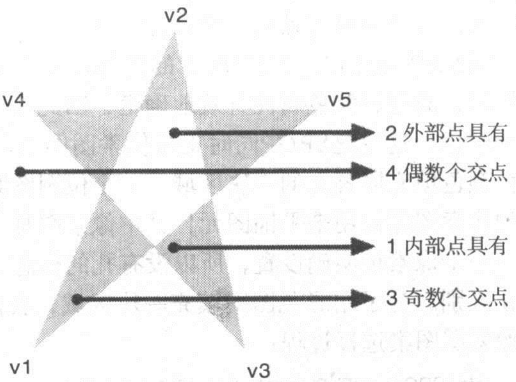


图2-9 确定多边形内部区域的奇偶规则

当绘图的时候，SRGP并不对每个点都进行这种测试；它利用第3章中介绍的优化多边形扫描技术，在这种技术中，奇偶规则能非常有效地运用于相邻的整行像素，无论这些像素是位于外部，还是位于内部。奇偶射线相交测试也可用于**关联拾取**（pick correlation），用来确定用户用光标选中的对象，这将在第7章中进行论述。

1. 区域填充类型和填充图案

利用以下函数，填充类型属性能用四种不同的方式来控制一个被填充图元内部的显示：

```
void SRGP_setFillStyle ( drawStyle SOLID / BITMAP_PATTERN_OPAQUE /
        BITMAP_PATTERN_TRANSPARENT / PIXMAP_PATTERN);
```

第一个选项SOLID利用当前的颜色值来产生一个均匀填充的图元（图2-8e，颜色被设成COLOR_WHITE）。接下来的两个选项BITMAP_PATTERN_OPAQUE和BITMAP_PATTERN_TRANSPARENT，用规则的非实心图案来填充图元，前一种用当前的颜色或者另一种颜色重写该图案F的所有像素（图2-8c），后一种用当前的颜色重写这个图元下面的某些像素，而让其他像素可透视（图2-8d）。最后一个选项PIXMAP_PATTERN，写图案时可包含任意数量的颜色，总是处于不透明的模式。

位图填充图案是一个以0和1为元素的位图矩阵，通过调用以下函数从可用的图案表中选取：

```
void SRGP_setFillBitmapPattern ( int patternIndex );
```

该图案表中的每一个条目中存储一个惟一的图案；由SRGP提供（参见参考手册）的图案表包括灰度色调（范围从黑色到白色）和各种各样的规则和随机图案。在透明方式中，这些图案通过如下方式产生。把图案表中的任何图案都当成一个小位图（比如8×8），在需要的时候通过重复拼贴来填充图元。在一个二值显示系统中，位图图案中值为1的地方显示当前颜色值，而值为0（孔）的地方保留原来颜色，从而使图案有一种透明的效果。这样，位图图案对透明方式中的图案起了“内存可写掩码”的作用，就像直线和边界图元的线型位掩码一样。

在更普遍应用的BITMAP_PATTERN_OPAQUE模式下，1置为当前颜色，但是，0置为另一种颜色，背景色事先通过以下函数设置：

```
void SRGP_setBackgroundColor ( int colorIndex );
```

在二值显示器上，OPAQUE方式下的每一种位图图案仅仅能产生两种不同的填充图案。例如，如果当前颜色被设置成黑色（背景色是白色），大部分为1的位图图案能在黑白显示器上产生暗灰色填充图案，如果当前颜色被设置成白色（背景色是黑色），则可产生亮灰色填充图案。在一个彩色显示器中，任一种前景和背景颜色的组合可以被用来实现两种色调的变化。二值显示系统中的一个典型应用程序通常在设置前景色的同时设置背景色，因为如果这两种颜色是一样的，不透明的位图模式是不可见的；应用程序可以生成一个SetColor过程，在系统设置前景色的时候，自动对照前景色来设置背景色。

图2-8是通过程序2-2中所示的代码片段产生的。具有双色调位图图案的优点是颜色不被明确规定，而通过颜色属性有效地确定，因此能由任何颜色组合生成。它的缺点是仅仅可以产生两种颜色，这也是SRGP同时支持像素图图案的原因。通常，我们喜欢采用一个明确的图案以多种颜色填充屏幕上的一块区域。一个位图图案是一个可以用来平铺图元的小位图，同样一个小的像素图能被用来平铺图元，其中像素图是一个指向颜色表索引的图案矩阵。因为在像素图中每一个像素被明确设置，所以没有孔的概念，因此在透明填充模式和不透明填充模式之间不存在区别。为了用彩色图案填充一块区域，我们选择PIXMAP_PATTERN填充类型并利用相关的像素图图案选择过程：

```
void SRGP_setFillPixmapPattern ( int patternIndex );
```

因为位图图案和像素图图案都是用索引到的当前颜色表中的颜色值来产生像素的颜色值，所以如果程序员改变了颜色表的内容，填充图元的结果将被改变。SRGP参考手册讨论了如何在位图和像素图图案表中改变和添加内容。虽然SRGP提供了位图图案表的默认内容，但它却没有给出一个默认的像素图图案表，因为有无数的彩色像素图图案都可能是有用的。

程序2-2 生成图2-8的代码

```
SRGP_setFillStyle(BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern(BRICK_BIT_PATTERN);            /* 砖形图案 */
SRGP_fillPolygon(3, triangle_coords);                    /* a */

SRGP_setFillBitmapPattern(MEDIUM_GRAY_BIT_PATTERN);      /* 50%灰度 */
SRGP_fillEllipseArc(ellipseArcRect, 60.0, 290);          /* b */

SRGP_setFillBitmapPattern(DIAGONAL_BIT_PATTERN);
SRGP_fillRectangle(opaqueFilledRect);                    /* c */

SRGP_setFillStyle(BITMAP_PATTERN_TRANSPARENT);
SRGP_fillRectangle(transparentFilledRect);               /* d */

SRGP_setFillStyle(SOLID);
SRGP_setColor(COLOR_WHITE);
SRGP_fillEllipse(circleRect);                            /* e */
```

29 ≀ 32

2. *应用屏幕背景*

在不透明的位图图案中，我们定义“背景色”为0位上绘制的颜色，但是术语背景的定义却不同。一般来说，用户希望在一个覆盖了不透明窗口或者整个屏幕的统一的*应用屏幕背景图案*上显示图元。应用屏幕背景图案经常是纯色0，即SRGP初始化屏幕后的颜色。然而，背景图案有时是非纯色的或者是其他纯色的；在这些情况下，在画任何其他图元之前，应用程序需要

通过绘制一个全屏大小的、用所需背景图案填充的矩形域，来完成应用程序背景设置。

一种常见的“擦去”图元的技术是以应用程序背景图案重画该图元，而不是每次图元被删掉之后都要重画整个图形。然而，当这个被擦去的图元与其他图元交迭时，这种“迅速和不干净的”更新技术会产生一个被损坏的图像。

例如，在图2-8中假设屏幕背景图案是纯白色，如果我们用纯白色重画标记为c)的矩形，则将在被填充的椭圆弧b)的下方留下一个缺口。“修补损坏的图形”需要回到应用程序数据库中重新定义图元（见习题2.8）。

2.1.4 存储和恢复属性

如前所述，SRGP对于各种图元支持多种属性。为了以后能恢复，私有属性可以被存储起来；这一功能对于设计那些无副作用（即不影响全局属性状态）的应用程序是很有用的。为方便起见，SRGP允许通过以下过程查询和恢复整个属性集——称为*属性组*：

```
void SRGP_inquireAttributes ( attributeGroup *group );
void SRGP_setAttributes ( attributeGroup group );
```

应用程序能访问SRGP定义的属性组（attributeGroup）记录的全部域，以便由查询函数返回的记录可以用于后面的选择性恢复。

2.1.5 文本

在图形软件包中定义和实现文本绘图往往是比较复杂的，因为文本包含大量的选项和属性，其中有字符的风格或**字体**（Times，Helvetica，**Bodoni**，等等）、外观（roman，**bold**，*italic*，underlined，等等）、尺寸（通常以磅[⊖]为单位）和宽度、字符间距、行距、画出字符的角度（水平，垂直，或者一指定的角度），等等。 33

在简单的软硬件环境中，最基本的字控制功能提供固定宽度和等间距字符，即所有字符占据同样的宽度，且字符之间的间距是常数。而另一方面，均衡间距技术则通过改变字符宽度和字符间距使得文本尽可能清晰和美观。书、杂志、报纸都用均衡间距，大多数的光栅显示器和激光打印机也是如此。SRGP提供折衷的功能：文本水平排列，字符宽度是变化的，但是字符间距是一定的。在均衡间距的简单形式下，应用程序可以注释图表，通过文本菜单和填充表格与用户交互，甚至执行简单的字处理。然而，基于文本的应用程序，如处理高质量文档的桌面排版系统，需要特殊的软件包以提供比SRGP更多的属性和规范控制。PostScript[ADOB85]提供了许多这样的先进功能，并且成为描述包含大量属性和选项的文本以及图元的工业标准。文本通过调用如下函数产生：

```
void SRGP_text ( point origin, char *text );
```

文本图元的位置通过定义它的*原点*（或称为**锚点**）来控制。原点的x坐标标记了第一个字符的左边界，y坐标确定了待显示字符串的基线。（**基线**是字符下方的一条假想线，如图2-10中所示的文本菜单按钮。一些字符，如“y”和“q”，其尾部（称为**字母下部**）延伸至基线的下方。）

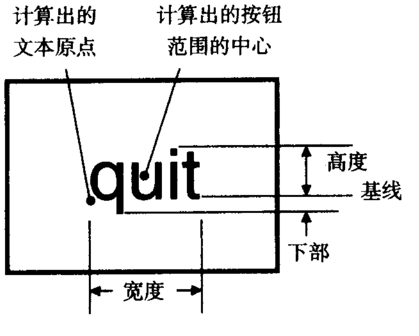


图2-10 将文本置于一个矩形按钮中央的尺寸以及利用这些尺寸计算出的用于居中的点

文本图元的外观仅由两个属性决定，即当前的颜色和字体，而字体是一个与实现相关的

⊖ 点（磅）是出版业中经常使用的一个单位；它大约等于1/72英寸。

不同尺寸和风格的字体表中的索引值：

```
void SRGP_setFont ( int valueIndex );
```

字体中的每一个字符都被定义为一个矩形位图，SRGP用字符的位图作为图案来填充一个矩形以绘制一个字符（在透明位图图案模式）。位图中的1定义字符的内部，0定义字符周围的空间和类似于字母“o”的中孔的字符内部间隙。（许多复杂的软件包用像素图的方式定义字符，允许对字符内部进行图案填充。）

格式化文本

由于SRGP提供有限的字体和尺寸功能，而且在不同硬件环境下的具体实现中很少提供等价的功能集，所以应用程序对于字符宽度和高度的控制是受限制的。为了将文本放置在一个平衡的位置（例如，在一个矩形框架内居中放置一个字符串），需要文本范围信息，所以SRGP提供了以下程序，利用字体属性的当前值查询一个给定字符串的范围：

```
void SRGP_inquireTextExtent( char *text, int *width, int *height, int *descent );
```

虽然SRGP不支持不透明位图模式来写字符，但是这样的模式是容易模仿的。举个例子，程序2-3中的函数显示了如何利用当前字体的范围信息和文本特有属性来产生黑色文本，并如图2-10所示居中放置于一个白色封闭的矩形内。这个过程首先创建一个指定尺寸并具有单独边界的背景矩形按钮，然后在它的内部居中显示文本。习题2.9是对于这个问题的一个变体。

程序2-3 生成图2-10中图形的代码

```
void MakeQuitButton( rectangle buttonRect )
{
   point centerOfButton, textOrigin;
   int width, height, descent;

   SRGP_setFillStyle(SOLID);
   SRGP_setColor(COLOR_WHITE);
   SRGP_fillRectangle(buttonRect);
   SRGP_setColor(COLOR_BLACK);
   SRGP_setLineWidth(2);
   SRGP_Rectangle(buttonRect);
   SRGP_inquireTextExtent("quit", &width, &height, &descent);
   centerOfButton.x = (buttonRect.bottomLeft.x + buttonRect.topRight.x)/2;
   centerOfButton.y = (buttonRect.bottomLeft.y + buttonRect.topRight.y)/2;
   textOrigin.x = centerOfButton.x – (width/2);
   textOrigin.y = centerOfButton.y – (height/2);
   SRGP_text(textOrigin, "quit" );
}
```

34
~
35

2.2 基本交互处理

既然我们知道了如何绘制基本的形状和文本，下一步就是学会如何编写利用键盘和鼠标等输入设备与用户有效通信的交互程序了。首先，我们看一看编写有效并且易于使用的交互程序的一般原则；然后，讨论逻辑（抽象）输入设备的基本注意事项。最后，我们将考察SRGP实现各种交互处理的机制。

2.2.1 人的因素

交互程序的设计者必须处理许多在非交互程序或批处理程序中不会出现的问题。这些问题被称为程序的**人的因素**，例如程序的交互风格（经常被称为“**视感**”）、易学性和易用性，它们

与程序功能的完整性和正确性同样重要。第8章将详细讨论展示良好的人的因素的人机交互技术。其中，主要的原则包括：

- 提供简单一致的交互序列。
- 不要以太多的选项和风格加重用户的负担。
- 在交互中的每一步清晰展示可选项。
- 给用户提供正确的反馈。
- 允许用户从错误中很好地恢复。

我们将在实例程序中遵守这些展现良好人的因素的技术原则。例如，用户通常以鼠标点击菜单的一个文本按钮来指出他想执行的下一个功能。同样通用的交互途径还有基本几何图元的调色板（图标菜单）、应用程序符号或者填充图案。菜单和调色板满足了前三个原则，因为它们列出了可用选项列表，并对这些选项提供了简单一致的选择方式。不可用的选项将被暂时删除，或者“灰显”，即以低亮度的灰度图案代替实色图案（见习题2.14）。

为了满足第四个原则，在菜单功能的每一步都提供反馈功能：应用程序会加亮菜单或对象选项（例如，把它放在一个矩形框内或利用相反的视觉显示）以引起用户注意。软件包本身也可以对输入设备的操作提供及时的“回应”。例如，当键盘输入时，在光标的位置字符迅速显现，随着鼠标在桌面上移动，光标回应在屏幕上相应的位置。图形软件包提供多样的光标形状以便 36
应用程序用它来反映当前的程序状态。在许多显示系统中，光标形状可以根据光标在屏幕上的位置动态改变。例如，在许多字处理程序中，在菜单区光标变成箭头形状，而在文本区变成闪烁的垂直线。

第五个原则，即从错误中很好地恢复，通常以“*cancel*”和“*undo/redo*”功能实现。这需要应用程序保持操作及其逆向、纠正动作的记录。

2.2.2 逻辑输入设备

1. SRGP中的设备类型

设计图形软件包的一个主要目标是实现设备无关性，它增强了程序的可移植性。为了实现图形输出的设备无关性，SRGP在抽象的整数坐标系统中定义图元，从而使帧缓存中独立像素的设置与应用程序脱离开来。同样，为提供图形输入抽象，SRGP支持一系列的**逻辑输入设备**使应用程序与物理输入设备的细节分离。SRGP支持的两种逻辑设备如下：

- **定位器**，定义屏幕坐标和一个或多个相应按钮状态的设备。
- **键盘**，字符串输入的设备。

SRGP将逻辑设备映射到可用的物理设备。例如，定位器可以映射为鼠标、游戏杆、输入板或触摸屏。这种从逻辑到物理的映射类似于传统的从过程语言到操作系统的映射，其中I/O设备，如终端、磁盘和磁带驱动设备被抽象为逻辑数据文件，这样既保证了设备无关性，又简化了应用程序设计。

2. 其他软件包中的设备处理

SRGP的输入模型实际上属于GKS输入模型和PHIGS输入模型的一部分。SRGP仅仅支持一种逻辑定位器和一种键盘设备，而GKS和PHIGS对每一种设备允许多种类型。这些软件包也支持其他的设备类型：**笔划**设备（返回由物理定位器输入的光标位置序列构成的折线），**选择**设备（将功能键输入板抽象化并返回一个键标识符），**定值器**（将一个滑块或控制转盘抽象化并返回一个浮点数），以及**拾取**设备（将一个带有选择按键的定位设备（如鼠标或数据输入板）抽象化，返回所选择的逻辑选项的标识）。其他程序包（如QuickDraw和X Window System）则

采用更加设备相关的方式控制输入设备，这样程序员可以对各个设备的操作进行更细的控制，
37 但是应用程序也因此变得更为复杂且降低了到其他平台的可移植性。

第8章将介绍逻辑设备的特性。这里，我们简单概括一下与逻辑设备交互的模式，然后再详尽讨论SRGP的交互过程。

2.2.3 采样与事件驱动处理

接收来自用户的信息有两种基本的方法。一种是**采样**（也称为**轮询**）方法，应用程序通过访问逻辑设备的当前值（称为该设备的**度量**）来执行下一步动作。不管自上一次采样后设备的度量是否发生变化，采样过程一直在执行；事实上，只有通过设备的连续采样才能让应用程序察觉设备状态的改变。对于交互式应用，这种方法是比较浪费的，因为在采样循环中要消耗大量CPU周期等待度量的变化。

为了避免CPU频繁的轮询循环，另一种办法是利用**中断驱动**进行交互；利用这种技术，应用程序使一种或多种设备可以输入，然后程序正常运行直到被某一输入**事件**（由用户动作引起的设备状态的改变）中断；接着，控制被异步传递到一个响应该事件的中断过程。对于每一个输入设备，都定义了一个**事件触发器**；事件触发器就是导致该事件发生的用户动作。通常触发器是一个按钮动作，如按下鼠标键（“mouse down”）或敲击键盘。

为了使应用程序员从繁杂而困难的异步传输控制中解脱出来，许多图形程序包（包括GKS、PHIGS和SRGP）提供**事件驱动**的交互过程，以同步方式模拟中断驱动交互。利用这种技术，应用程序在启用设备后继续执行；在程序执行的同时，由软件包监控设备并把每一个事件的信息存储于一个事件队列中（图2-11）。应用程序可在方便的时候随时检查事件队列并按时间顺序处理这些事件。实际上，这等于让应用程序自己决定被中断的时刻。

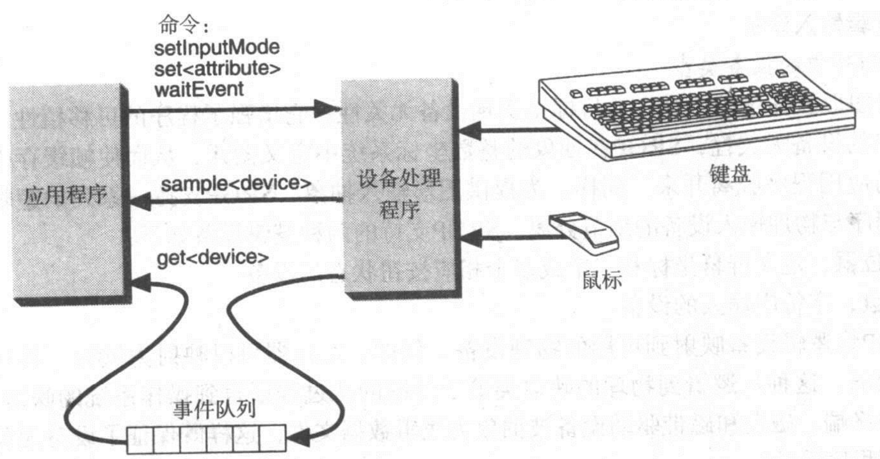


图2-11 采样与使用事件队列的事件处理

当应用程序检查事件队列的时候，它决定是否进入等待状态。如果队列中包括一个或多个事件报告，应用程序取出头事件（代表最早发生的事件）并读取其信息。如果队列是空的且下一个状态不是等待状态，应用程序将被告知无可用事件，并继续原来的执行。如果队列是空的并且即将进入等待状态，应用程序将暂停，直到下一个事件发生或等待事件超过程序指定的最长等待时间。事实上，事件模式用效率更高的事件队列等待代替了输入设备的轮询。

总之，在采样模式中，不管是否有用户动作，都要轮询设备并搜集事件度量。在事件模式中，应用程序可以获得用户动作的事件报告或等待直到有用户动作（或超时）发生。这种“仅

当用户有动作时才反应”的事件模式的行为特征是事件驱动输入与采样输入的最重要的不同点。事件驱动编程看起来可能比采样更复杂，但是你可能已经对一个类似的技术比较熟悉了，如C语言程序中使用的scanf函数：C启用了键盘，应用程序在scanf函数中等待用户完成一行文本的输入。有些环境允许使用scanf语句访问scanf被发出之前键入和排队的字符。

SRGP或类似软件包中的简单的事件驱动程序遵循1.4.3节中介绍并由程序2-4中的伪代码描述的“乒乓”交互。这种交互能够被很好地建模为一个有限状态自动机。允许程序和用户动作同时发生的更多复杂类型的交互，将在第8章中讨论。

程序2-4 事件驱动的交互模式

```
初始化，包括生成初始图像；
激活交互设备使之处于事件模式；
do {      /* 主事件循环 */
   等待任何设备上的用户触发事件；
   switch(引发事件的设备) {
      case DEVICE_1: collect DEVICE_1 事件度量数据、处理、响应；
      case DEVICE_2: collect DEVICE_2 事件度量数据、处理、响应；
      ...
   }
}
while (用户没有请求退出);
```

事件驱动的应用程序大部分时间处在等待状态，因为交互过程主要是由用户决定下一步行动的“思考时间”控制的；即使在快节奏游戏程序中，一秒钟里用户所能产生的事件数量也只是应用程序所能处理的一小部分。由于SRGP主要使用实（硬件）中断来实现事件模式，等待状态并不占用CPU时间。在多任务系统中，这一优点是明显的：事件模式的应用程序只需要 38~39
CPU完成用户动作发生后短暂的突发活动，因此可以腾出CPU时间给其他任务。

还要注意一点，就是关于事件模式的正确使用。虽然队列机制允许程序和用户交互异步进行，但是用户不允许比程序超前太多，因为每一事件都会产生一些回应和来自应用程序的一些反馈。有经验的用户在系统正在处理较早的请求时，会“**提前键入**”文件名甚至操作系统命令等参数，特别是在字符回应很快的时候。相反，“**提前鼠标操作**”对于图形命令毫无用处（而且非常危险），因为用户通常都需要观看更新后的屏幕以获取应用模型的当前状态，才能进行下一步图形交互。

2.2.4 采样模式

1. 激活、关闭和设置设备模式

以下过程可以激活或关闭设备；它以设备和模式作为参数：

```
void SRGP_setInputMode ( inputDevice LOCATOR / KEYBOARD,
     inputMode INACTIVE / SAMPLE / EVENT);
```

这样，为了把定位器设为采样模式，我们调用

```
SRGP_setInputMode (LOCATOR, SAMPLE);
```

最初，两个设备都处于非激活状态。设置一个设备为某种模式绝不会影响其他输入设备——即使两者被同时激活，也不一定要在同一模式下工作。

2. 定位器的度量

定位器是鼠标或数据输入板的逻辑抽象，返回光标的屏幕坐标（x, y）、最近发生变化的按钮的编号以及以*chord*数组形式返回按钮的状态（因为能同时按下多个按钮）。其中第二个域可

以让应用知道是哪个按钮触发了那一事件。

```
typedef struct {
    point  position;
    enum {
        UP, DOWN
    } buttonChord[MAX_BUTTON_COUNT];          /* 通常是1~3 */
    int buttonOfMostRecentTransition;
} locatorMeasure;
```

在使用SRGP_setInputMode过程以采样模式激活定位器后，我们就可通过调用以下过程获取定位器的当前度量：

40
```
void SRGP_sampleLocator ( locatorMeasure *measure );
```

让我们看一下程序2-5中的采样应用程序原型：一个简单的仅使用定位器上按钮1的“着色”循环。这个着色过程会沿着用户按下该按钮同时挪动定位器的轨迹着色；定位器在用户移动时被循环采样。首先，我们必须对定位器按钮进行采样，直到它被按下，从而确定用户开始着色；然后，我们对定位器进行采样，并在每个采样点上着色（在这个例子中是一个填充的矩形），直到用户释放这个按钮。

程序2-5 着色的采样循环

```
设置颜色/图案属性，并设画刷尺寸为halfBrushHeight和halfBrushWidth;
SRGP_setInputMode(LOCATOR, SAMPLE);

/* 首先，一直采样直到按钮按下 */
do {
    SRGP_sampleLocator(locMeasure);
} while (locMeasure.buttonChord[0] == UP);

/* 执行画图循环：
   不断地放置画刷然后采样，直到按钮抬起 */
do {
    rect = SRGP_defRectangle( locMeasure.position.x – halfBrushWidth,
                    locMeasure.position.y – halfBrushHeight,
                    locMeasure.position.x + halfBrushWidth,
                    locMeasure.position.y + halfBrushHeight );
    SRGP_fillRectangle (rect );
    SRGP_sampleLocator( &locMeasure );
} while ( locMeasure.buttonChord[0] == DOWN );
```

这个过程的结果是粗糙的：着色的矩形不是过于紧凑，就是过于分散，它们的密度完全依赖于定位器在两个连续采样之间移动的距离。采样速率主要由CPU运行操作系统、软件包和应用程序的速度决定。

虽然两种逻辑设备都可以使用采样模式，但是键盘设备几乎总是在事件模式下运作，所以这里没有提及关于它的采样技术。

2.2.5 事件模式

1. 使用事件模式启动采样循环

虽然着色例子中的两个采样循环（一个检测按钮按下，另一个完成着色直到按钮松开）都能运行，但是它们无谓地增加了CPU的工作量。虽然这种方法在个人计算机上可能不会产生严重的问题，但是在一个多任务系统中却是不适宜的，更不用说要实现分时了。尽管对定位器重

复采样的着色循环是必不可少的（因为我们需要知道按钮按下时定位器在任何时间的位置）， 41
我们并不需要使用采样循环等待启动着色过程的按钮按下事件。在等待事件发生而不需要度量信息的时候，我们可以采用事件模式。

2. SRGP_waitEvent

当SRGP_setInputMode在事件模式下激活一个设备后，程序可以通过调用以下过程进入等待状态来检测事件队列：

```
inputDevice SRGP_waitEvent ( int maxWaitTime );
```

如果队列非空，这个过程立即返回；否则，第一个参数将指定这个过程在队列为空时的最大等待时间（以1/60秒计）。负的*maxWaitTime*值（由符号常量INDEFINITE定义）将导致过程无限期等待，0值导致过程立即返回，不论队列状态如何。

发生头事件的设备标识在*device*参数中返回。如果指定的时间内没有事件发生，即如果设备超时，返回特殊值NO_DEVICE。接下来可以通过检测设备以决定如何提取头事件的度量（将在本节后面描述）。

3. 键盘设备

键盘设备的触发事件取决于键盘设备所在的**处理模式**。在EDIT模式中，应用程序接收用户输入的字符串（例如，文件名和命令），用户在键入并编辑字符串后按下回车键触发事件。在RAW模式的交互过程中，键盘受到紧密监控，每按下一个键就触发一个事件。应用程序可以使用以下过程设置处理模式：

```
void SRGP_setKeyboardProcessingMode ( keyboardMode EDIT / RAW );
```

在EDIT模式中，用户可以键入完整的字符串，需要时可用退格键改正，最后使用回车键（或输入键）作为触发器。这种模式适用于键入完整的字符串，如文件名或图形标签。除了退格和回车键之外所有的控制键都被忽略，因为字符串出现在触发的时刻，所以度量就是字符串本身。然而，在RAW模式中，每一个键入的字符，包括控制字符，都是一个触发器并作为度量单独返回。这种模式在单个键盘字符作为命令（例如，移动光标、简单的编辑操作或者视频游戏）时使用。RAW模式不提供回应，而EDIT模式回应字符串并显示在屏幕上，同时在下一文本输入位置显示一个**文本光标**（如下划线或块字符）。每按一次退格键文本光标回撤一格并删掉一个字符。

在SRGP_waitEvent返回设备代号KEYBOARD后，应用程序通过调用以下过程获得与该事件相关的度量：

```
void SRGP_getKeyboard (char *measure , int buffersize);
```
42

当键盘设备在RAW模式下被激活后，它的度量保持一个字符的长度。在这种情况下，度量字符串的第一个字符作为RAW模式的度量返回。

从程序2-6所示的程序段可以看到EDIT模式的使用方法。它从用户端接收一系列文件名，同时删除相应的文件。如果用户输入一个空字符串（不按任何其他字符直接按回车键），交互结束。在交互过程中，程序无限期等待直到用户输入下一个字符串。

虽然这段代码明确地指定了文本提示符的显示位置，但是它并没有指定用户键入字符串（以及用退格键修正字符串）的位置。键盘回应的位置是由程序员指定的，我们将在2.2.7节中讨论这个问题。

程序2-6 EDIT模式下的键盘交互

```
#define KEYMEASURE_SIZE 80
SRGP_setInputMode(KEYBOARD, EVENT);      /* 假设只有键盘激活 */
SRGP_setKeyboardProcessingMode(EDIT);
pt = SRGP_defPoint( 100, 100 );
SRGP_text( pt, "Specify one or more files to be deleted; to exit press Return\n" );

/* 主事件循环 */
do {
   inputDev = SRGP_waitEvent( INDEFINITE );
   SRGP_getKeyboard( measure , KEYMEASURE_SIZE );
   if (strcoll(measure, "" ))
      DeleteFile(measure);              /* DeleteFile做确认等 */
}
while ( strcoll(measure, "" ) );
```

4. 定位器设备

定位器设备的触发事件是按下或松开鼠标键。在SRGP_waitEvent返回设备代号LOCATOR后，应用程序通过调用以下过程获得与该事件相关的度量：

```
void SRGP_getLocator ( locatorMeasure *measure );
```

通常，度量的*position*字段用来确定用户指定的点在屏幕上的位置。例如，如果定位器光标位于一个显示菜单按钮的矩形域中，事件将被视为某一动作的请求；如果光标位于主绘画区域中，事件则可以表示选中某一个已经存在的图形对象或是放置一个新的图形对象的位置，这取决于光标点在图形对象的内部还是外部。

程序2-7所示的伪代码（与前面所示的键盘伪代码类似）实现了定位器的另一个用途，即让用户指定放置标记的位置。当光标指向一个屏幕按钮（一个标有“quit”的矩形域）时，按

43 下定位器按钮就可以退出标记定位循环。

程序2-7 定位器交互

```
#define QUIT 0
在屏幕上创建退出按钮;
SRGP_setLocatorButtonMask( LEFT_BUTTON_MASK );
SRGP_setInputMode( LOCATOR, EVENT );        /* 假设只有定位器激活 */
/* 主事件循环 */
terminate = FALSE;
do {
   inputDev = SRGP_waitEvent( INDEFINITE );
   SRGP_getLocator( &measure );
   if (measure.buttonChord[QUIT] == DOWN ) {
      if PickedQuitButton( measure.position ) terminate = TRUE;
      else
         SRGP_marker( measure.position );
   }
}
while ( !terminate );
```

在这个例子中，只有定位器按钮1的按下事件有意义；按钮的释放被忽略。注意在下一个按按钮事件发生之前定位器按钮必须松开——这个事件是通过转变触发的，而不是通过按钮的状态来触发。此外，为保证交互过程不受其他按钮事件的干扰，应用程序调用以下函数通知SRGP哪个按钮将触发定位器事件：

```
void SRGP_setLocatorButtonMask ( int activeButtons );
```

按钮屏蔽位的值为LEFT_BUTTON_MASK、MIDDLE_BUTTON_MASK和RIGHT_BUTTON_MASK。组合屏蔽位由各个值的逻辑“或”形成。默认的定位器按钮屏蔽位为1，但无论屏蔽位是几，每个按钮都有自己的度量。在支持少于三个按钮的实现中，指向任何不存在按钮的引用都会被SRGP忽略，而这些按钮的度量都包含UP。

函数PickedQuitButton将度量位置与退出按钮矩形框的边界进行比较，并返回一个布尔值来表示用户是否选中退出按钮。这个过程是一个简单的**关联拾取**（pick correlation）的例子，我们将在2.2.6节讨论。

5. 等待多重事件

程序2-6和程序2-7所示的代码段并没有说明事件模式最大的优越性：同时等待多个设备的能力。SRGP将来自启用设备的事件按时间顺序放入队列，应用程序调用SRGP_waitEvent过程从队列中取出第一个事件。与硬件中断不同，后者以优先权顺序处理中断，而前者严格按照时间顺序处理事件。应用程序通过检查返回的设备代号来确定哪个设备引发了该事件。

程序2-8中所示的过程允许用户将任意多个小圆形标记放置在矩形绘图区内的任意位置。用户把光标挪到想要的位置，按下键1就可放置一个标记；用户可以按下按钮3或输入“q”或“Q”字符终止交互。 44

程序2-8 同时使用多个设备

```
#define PLACE_BUTTON 0
#define QUIT_BUTTON 2

生成初始的屏幕布局;
SRGP_setInputMode( KEYBOARD, EVENT );
SRGP_setKeyboardProcessingMode( RAW );
SRGP_setInputMode( LOCATOR, EVENT );
SRGP_setLocatorButtonMask( LEFT_BUTTON_MASK | RIGHT_BUTTON_MASK );
                                        /* 忽略第2个按钮 */
/* 主事件循环 */
terminate = FALSE;
do {
   device = SRGP_waitEvent( INDEFINITE );
   switch (device ) {
      case KEYBOARD:
         SRGP_getKeyboard( keyMeasure, lbuf );
         terminate = (keyMeasure[0] == 'q') || (keyMeasure[0] == 'Q');
         break;
      case LOCATOR: {
         SRGP_getLocator( &locMeasure );
         switch ( locMeasure.buttonOfMostRecentTransition ) {
            case PLACE_BUTTON:
               if (( locMeasure.buttonChord[PLACE_BUTTON] == DOWN )
                  && InDrawingArea( locMeasure.position ))
                     SRGP_marker( locMeasure.position );
               break;
            case QUIT_BUTTON:
               terminate = TRUE;
               break;
         } /* 按钮情况 */
      }  /* 定值器情况 */
   }  /* 设备情况 */
}
while (!terminate );
```

2.2.6 交互处理中的关联拾取

图形应用程序通常将屏幕划分为一些完成特定功能的区域。当用户点击定位器按钮时，应用程序必须准确地判断用户选择了哪个屏幕按钮、图标或其他对象，以做出适当的响应。这个判断过程称为**关联拾取**，它是交互式图形应用的基本组成部分。

使用SRGP的应用程序通过确定光标位于哪个区域，以及位于该区域内的哪个对象上来实现关联拾取。空区域内的点将被忽略（例如，菜单上菜单按钮之间的点），或用于指定放置新
45 对象的位置（如果点位于主绘图区域内）。因为屏幕上许多区域都是正矩形，所以几乎所有与关联拾取有关的工作都可以由一个简单的、常用的布尔函数完成，这个函数判定给定的点是否在给定的矩形内部。随SRGP发布的GEOM软件包提供了这个函数（GEOM_ptInRect）以及其他坐标运算工具。（有关关联拾取的更详细的资料，参见7.11.2节。）

让我们看一个关联拾取的经典例子。现有一绘图应用程序，程序中屏幕顶端设有一**菜单条**。菜单条中包含各下拉菜单的名字，称为菜单**标题**。当用户选取一个标题时（将光标放置在显示标题的文字串上并按下定位器键），相应的**菜单体**显示在屏幕上标题的下方，同时标题高光显示。当用户选择菜单体上的一项后（释放定位器按钮），菜单体消失，标题恢复正常显示。屏幕其他部分是供用户放置和拾取对象的主绘图区。应用程序在创建每个对象时，分配给该对象一个惟一的标识符（ID），由关联拾取过程返回并进行进一步的处理。

通过按下按钮事件从定位器得到一个点后，程序按照程序2-9所示的过程进行高层交互处理；本质上这是一个调度过程，它根据关联拾取在菜单条中还是在主绘图区中分别调用菜单拾取过程和对象拾取过程。首先，如果光标在菜单条内，会有一个辅助相关过程判断用户是否选择了一个菜单标题。如果是，将调用一个过程（细节见2.3.1节）来执行菜单交互；同时返回菜单体中被选中的项的下标（如果有的话）。菜单ID和项目下标在一起可以惟一确定响应操作。如果光标不在菜单条内而是在主绘图区内，另一个辅助相关过程将被调用以确定被拾取的对象（如果有的话）。如果一个对象被拾取，程序会调用一个处理过程做出正确的响应。

程序2-9 菜单处理的高层交互概要

```
void HighLevelInteractionHandler( locatorMeasure measureOfLocator )
{
   if ( GEOM_pointInRect( measureOfLocator.position, menuBarExtent ) {
        /* 找出用户选择哪个菜单头（如果有）;
           然后，到菜单体 */
      menuID = CorrelateMenuBar( measureOfLocator.position );
      if ( menuID > 0 ) {
         chosenItemIndex = PerformPulldownMenuInteraction( menuID );
         if (chosenItemIndex > 0 )
            PerformActionChosenFromMenu(menuID, chosenItemIndex);
      }
   else        /* 用户在绘图区域内拾取，找出内容和响应 */
   {
      objectID = CorrelateDrawingArea( measureOfLocator.position );
      if ( objectID > 0 ) ProcessObject( objectID );
   }
}
```

46

CorrelateMenuBar过程通过对菜单条内的每个菜单标题调用一次GEOM_pointInRect实现了较好的关联拾取；它访问的是一个存储各标题矩形屏幕区的数据库。CorrelateDrawingArea 过程的关联拾取过程更加复杂，因为通常绘图区内的对象会重叠，而且不一定都是矩形的。

2.2.7 设置设备度量和属性

各输入设备都有其自己的属性集合，应用程序可以通过设置这些属性来定制反馈给用户的信息。（前面提到的按钮屏蔽位也是一个属性；它与这里介绍的属性的不同之处在于它不影响反馈。）与输出图元属性类似，输入设备属性由特定的过程模态地设置。设备属性可以随时设置，无论设备是否处于激活状态。

另外，各输入设备的度量（通常都由用户的操作决定）也可由应用程序来设置。与输入设备的属性不同，当输入设备被关闭时，该设备的度量复位为默认值；这样，设备在被重新激活时就有可预知的值，对编程者和用户都很方便。这种自动复位功能可以由显式设置设备的度量替代。

1. 定位器回应属性

定位器可以使用多种回应类型。程序员可以调用函数

```
void SRGP_setLocatorEchoType ( echoType NO_ECHO / CURSOR /
    RUBBER_LINE /RUBBER_RECT );
```

同时控制回应类型和光标形状。参数的默认值为CURSOR，SRGP软件包提供一个光标表供应用程序选择所需的光标形状（参阅参考手册）。动态指定光标的形状通常用于根据光标所在区域改变光标的形状以提供反馈。RUBBER_LINE和RUBBER_RECT回应通常用来指定一条直线或一个框。当用户移动定位器时，SRGP通过设置这两个属性自动绘制连续更新的直线或矩形。直线或矩形是由两个点，即锚点（定位器的另一属性）和当前的定位器位置来定义的。图2-12说明了如何使用这两种模式定义直线和矩形。

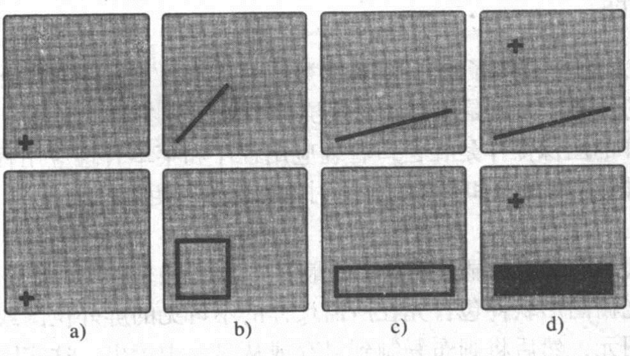


图2-12 Rubber-echo图解。a) 按下按钮，启动回应；b) 橡皮图元回应定位器；c) 定位器位置返回给应用程序；d) 应用程序画直线，并恢复回应

在图2-12a中，光标是个十字游标，用户正要按下定位器按钮。应用程序启动橡皮回应（rubber echo）响应按下按钮的动作，锚点定在当前定位器的位置。在图2-12b和图2-12c中，程序以橡皮图元回应定位器的移动。当用户释放按钮时，图2-12c中的定位器位置返回给应用程序，而应用程序则画出相应的线图元或矩形图元，并恢复正常的光标形状（见图2-12d）。

橡皮回应的锚点是通过调用以下函数设置的：

```
void SRGP_setLocatorEchoRubberAnchor ( point position );
```

应用程序通常将从最近一次按定位器按钮事件获得度量的position域指定为锚点的位置，因为按键动作通常触发橡皮回应序列。

2. 定位器度量控制

一旦定位器被关闭，定位器度量的position值自动复位到屏幕的中心。除非程序员重新设

置，否则度量（如果回应被激活，还包括反馈位置）在设备被重新激活时将被初始化到相同的位置。任何时候，无论设备正在工作还是不在工作，编程者可以调用函数

```
void SRGP_setLocatorMeasure ( point position );
```

重置定位器的度量（*position*部分，而不是与按键有关的域）。

当定位器处于关闭状态时重置度量，结果不会立刻在屏幕上反映出来；但当定位器处于激活状态时重置度量，会及时改变相应的回应（如果有的话）。这样，如果程序中要让光标在定位器激活时位于除中心点以外的其他位置，必须在调用SRGP_setInputMode前以所需初始位置调用SRGP_setLocatorMeasure函数。这一技术通常用于实现光标位置的连续性：定位器关闭前的最后一个度量被存储起来，当设备被重新激活时，光标可以回到该位置。

3. 键盘属性和度量控制

定位器回应的是物理设备的运动位置，而键盘设备的回应在屏幕上没有确定的位置。因此，位置作为键盘设备的属性之一（默认值与具体实现有关），可通过调用以下函数设置：

47~48

```
void SRGP_setKeyboardEchoOrigin ( point origin );
```

当键盘设备关闭时，键盘的默认度量自动复位为空串。如果在激活键盘前将度量设置为一个非空的初始值，可以很方便地实现一个默认输入串的显示（在回应一开始就由SRGP显示），用户可以接受它或在修改后按回车键，从而减少击键次数。键盘的度量是通过调用以下函数设置的：

```
void SRGP_setKeyboardMeasure ( char *measure );
```

2.3 光栅图形特性

到现在为止，我们已经介绍了SRGP的大多数特性。这一节将讨论其余的功能，它们充分利用了光栅显示硬件的优势，尤其是保存和恢复屏幕中被其他图像（例如，窗口或临时菜单）覆盖区域的能力。这样的图像操作是在窗口管理应用程序和菜单管理应用程序的控制下完成的。我们还引入屏外位图来存储窗口和菜单，并将讨论如何进行矩形框裁剪。

2.3.1 画布

让复杂的图标或菜单快速地显示或消失的最好方法是在内存中创建它们，然后将它们按需要复制到屏幕上。光栅图形软件包首先在所需尺寸的不可见的屏外位图或像素图（SRGP中称为的**画布**）中生成图元，然后将画布复制到显存或从显存中读出。这实际上是一种缓冲技术。使用了我们即将讨论的快速SRGP_copyPixel操作后，整块地来回移动像素一般要比重新生成信息更快。

SRGP*画布*是一个将图像作为2D像素数组存储的数据结构。它也存储一些有关图像大小和属性的控制信息。每个画布在各自的笛卡儿坐标系下显示图像，这与图2-1中所示的屏幕显示相同；实际上，屏幕本身就是个画布，特殊之处仅在于它是惟一被显示的画布。要显示一个存储在屏外的画布上的图像，应用程序必须将它复制到屏幕画布上。在新图像（例如菜单）即将显示的屏幕部分内的图像，其像素将预先复制到其他屏外画布上存储起来。待菜单选中之后，这些像素再从画布复制回以恢复原图像。

在任何时候，只有一个当前活动画布，新的图元将绘制到这个画布上，而新的属性设置也将对它生效。该画布可能是屏幕画布（我们使用的默认画布）或是一个屏外画布。传给图元过
49 程的坐标是以当前活动画布的局部坐标空间形式表示的。每个画布都有自己完整的SRGP属性集，这些属性影响所有存储在该画布上的图形，而且当创建画布时，这些属性被设置为标准的

默认值。对属性设置过程的调用仅修改当前活动画布的属性。为方便起见，可以把画布看成一个虚拟屏幕，它具有程序指定的尺寸、与自己关联的像素图、坐标系和属性组。画布的这些特性有时也被称为是画布的**状态**或**内容**。

当SRGP被初始化时，**屏幕画布**自动创建并被激活。我们讨论过的所有程序只能在该画布内生成图元。它是惟一在屏幕上可见的画布，而且其ID是SCREEN_CANVAS，为SRGP常量。通过调用以下过程可创建一个新的屏外画布，该过程返回分配给新画布的ID：

```
int SRGP_createCanvas ( int width, int height );
```

与屏幕类似，新画布的局部坐标系的原点（0, 0）在左下角，右上角位于（*width*−1, *height*−1）。一个1 × 1的画布定义的宽和高均为1，它的左下角和右上角均在（0, 0）！这与我们对网格交点处的像素的处理相一致：在1 × 1的画布上的单个像素位于（0,0）。

新创建的画布被自动激活，并且其像素初始化为颜色0（在任何图元显示前，屏幕画布也完成同样操作）。一旦画布被创建，它的尺寸不能再改变。同样，由于SRGP使用的像素的位数与硬件要求一致，所以程序员也不能控制画布中各像素所占的位数。画布的属性作为其“局部”状态信息的一部分被保存；这样，在创建一个新的活动画布之前，程序不需要显式地保存当前活动画布的属性。

应用程序通过调用以下函数选择一个已经创建的画布作为当前活动画布：

```
void SRGP_useCanvas ( int canvasID );
```

画布被激活决不意味着该画布变为可见；屏外画布上的图像必须拷贝到屏幕画布上（使用刚才提到的SGRP_copyPixel过程）。

可以通过调用以下过程删除画布：

```
void SRGP_deleteCanvas ( int canvasID );
```

但该过程不能用来删除屏幕画布或当前活动画布。以下过程允许查询画布的尺寸；其中一个返回定义画布坐标系（左下角点通常为（0,0））的矩形，另一个将宽和高分别作为独立量返回。

```
rectangle SRGP_inquireCanvasExtent ( int canvasID );
void SRGP_inquireCanvasSize ( int canvasID, *width, *height );
```

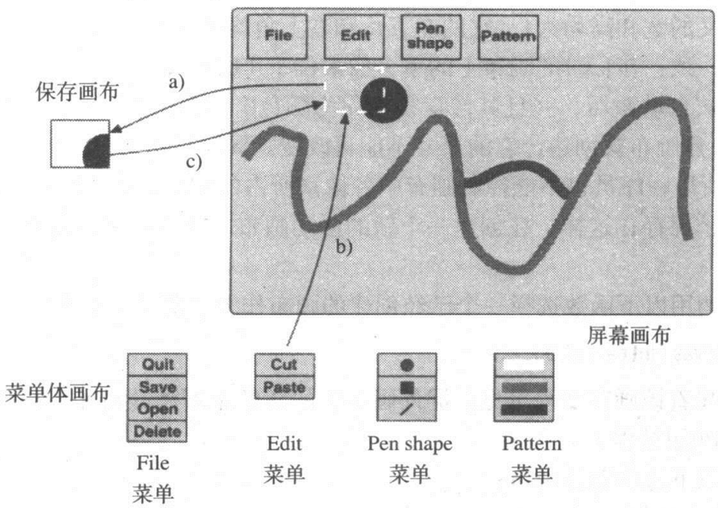
让我们来看如何使用画布实现在程序2-9和2.2.6节中提到的高层交互处理程序调用的Perform PulldownMenuInteraction过程。该过程由程序2-10给出的伪代码实现，图2-13展示了它 50
的动作序列。每个菜单有各自惟一的ID（由CorrelateMenuBar函数返回），它可用来定位包含以下有关菜单体外观信息的数据库记录：

- 存储菜单体的画布的ID。
- 用屏幕画布坐标来表示的矩形区（伪代码中称为*menuBodyScreenExtent*），即当用户点击菜单标题下拉菜单时，菜单体的显示区域。

程序2-10 PerformPulldownMenuInteraction的伪代码

```
int PerformPulldownMenuInteraction( int menuID );
/* 画布矩形区域的保存/复制在后面描述 */
{
  高光显示菜单条中的菜单头;
  menuBodyScreenExtent = 菜单体出现的屏幕区域矩形;
   在临时画布上保存menuBodyScreenExtent的当前像素;
     /* 参见图2-13a */
```

```
    将菜单体图像从体画布复制到menuBodyScreenExtent;
      /* 参见图2-13b和程序2-11中的C代码 */
    等待按钮弹起，提醒用户做出选择，然后得到定位器的度量；
    将临时画布上保存的图像复制到menuBodyScreenExtent;
      /* 参见图 2-13c */
    if ( GEOM_pointInRect(measureOfLocator.position, menuBodyScreenExtent )
      使用度量位置的y坐标计算并返回所选项的索引;
    else
      return 0;
}
```



51

图2-13 保存和恢复被菜单体覆盖的区域

2.3.2 矩形框的裁剪

通常，为了保护画布的其他区域，需要把图形图元的作用限制在活动画布的一个子区域内。SRGP通过设置**裁剪矩形**（clip rectangle）属性实现该功能。所有的图元都被该矩形的边界裁剪；也就是说，落在裁剪矩形外的图元（或图元的一部分）不画出来。与其他属性一样，裁剪矩形属性可在任何时候改变，它最近一次的设置存储在画布的属性组中。默认的裁剪矩形（我们至今使用的）是整个画布；它可以变得比画布小，但不能扩大超出画布的边界。裁剪矩形属性的相关设置与查询调用如下：

```
void SRGP_setClipRectangle ( rectangle clipRect );
rectangle SRGP_inquireClipRectangle ();
```

如2.2.4节中介绍的绘图应用程序就可以利用裁剪矩形把着色的位置限制在屏幕的绘图区内，从而保证周围的菜单区域不受影响。尽管SRGP仅提供了一个正矩形裁剪边界，一些更高级的软件（如PostScript）则提供了多重、任意形状的裁剪区。

2.3.3 SRGP_copyPixel操作

强大的SRGP_copyPixel命令是典型的光栅命令，用硬件直接实现时经常被称为bitBlt（位块传输）或pixBlt（像素块传输）；在20世纪70年代早期，Xerox Palo Alto研究中心在开拓ALTO位图工作站上首先实现了其微代码[INGA81]。该命令用来从画布的矩形区域（源区域）

将一个像素数组拷贝到当前活动画布的目的区域（见图2-14）。SRGP仅提供有限的功能，即目的矩形必须与源矩形具有相同的尺寸。在更强大的版本中，源区域可以自动变换尺寸并被拷贝到一个与它大小不同的目的区域。同时还有一些额外的功能，例如**屏蔽**（mask）功能可以选择性地屏蔽源或目的像素拷贝，而**半色调图案**（halftone pattern）可以用来“筛掩”（即浓淡遮蔽）目的区域。

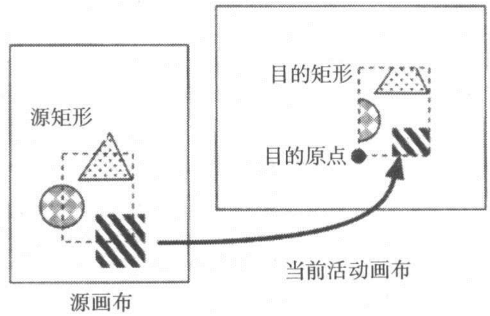


图2-14 SRGP_copyPixel

SRGP_copyPixel可以在任何两个画布间拷贝，其调用方法如下：

```
void SRGP_copyPixel ( int sourceCanvasID, rectangle sourceRect,
    point destCorner );
```

*sourceRect*指定任意画布中的源区域，*destCorner*定义位于当前活动画布中的目的矩形的左下角，它们都以各自的坐标系表示。为了防止图元在受保护的区域内生成像素，拷贝操作同样受到活动画布上裁剪矩形的约束。这样，像素最终拷贝到的区域是目的画布，目的区域和裁剪矩形三者的交集，如图2-15中的条纹区域所示。

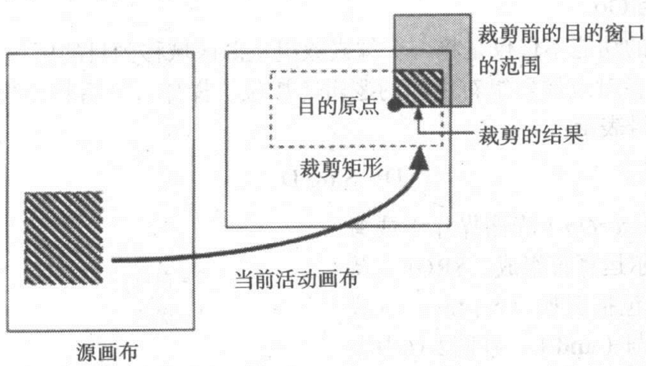


图2-15 copyPixel中的裁剪

为了介绍如何使用copyPixel处理下拉菜单，我们执行PerformPulldownMenuInteraction函数（程序2-10）中的第4句伪代码——“copy menu body image（拷贝菜单体图像）”。在该伪代码的第3句，我们已经把菜单体将要显示的屏幕区域存储在一个屏外画布中；现在，我们希望将菜单体复制到屏幕上。

C代码如程序2-11所示。我们必须区别两个大小相同但用不同的坐标系表示的矩形。第一个矩形，我们在代码中称之为*menuBodyExtent*，就是菜单体的画布在其坐标系中的区域。该区域用来作为SRGP_copyPixel操作的源矩形，将菜单放到屏幕上。*menuBodyScreenExtent*是一个大小相同的矩形，它在屏幕坐标中指定菜单体将要显示的位置；该区域的左下角是与菜单标题的左边水平对齐的，其右上角紧靠菜单条的底部。（图2-13用点划线框表示Edit菜单的屏幕区域，它的菜单体区域用实线表示。）*menuBodyScreenExtent*左下角点用来指定拷贝菜单体的SRGP_copyPixel的目的位置（图2-15）。另一方面，它也用于存储要被菜单体覆盖的屏幕区域的源矩形和最后恢复操作的目的矩形。 52~53

程序2-11 将菜单体拷贝到屏幕上的代码

```
/* 这段代码将菜单体的图像拷贝到屏幕上，
   其在屏幕上的位置由存放在菜单体的记录决定 */

/* 保存当前活动画布（不必是屏幕）的ID */
saveCanvasID = SRGP_inquireActiveCanvas();

/* 保存当前屏幕画布的裁剪矩形的属性值 */
SRGP_useCanvas( SCREEN_CANVAS );
saveClipRectangle = SRGP_inquireClipRectangle();

/* 临时设置屏幕剪切矩形，允许向整个屏幕上写 */
SRGP_setClipRectangle( SCREEN_EXTENT );

/* 将菜单体从它的画布中复制到菜单栏上的标题下面的正确区域 */
SRGP_copyPixel( menuCanvasID, menuBodyExtent,
    menuBodyScreenExtent.bottomLeft );

/* 恢复屏幕属性和活动画布 */
SRGP_setClipRectangle( saveClipRectangle );
SRGP_useCanvas( saveCanvasID );
```

请注意保存和恢复应用程序的状态以消除副作用。在拷贝前，我们应将屏幕裁剪矩形设置为SCREEN_EXTENT，或者我们可以把它设为*menuBodyScreenExtent*的具体值。

2.3.4 写模式或RasterOp

SRGP_copyPixel的功能不仅仅是将一个像素数组从源区域移到目的区域。它也可以执行源区域和目的区域内的各对应像素组对之间的逻辑（按位）操作，然后将结果放到目的区域中。该操作可以用以下符号表示：

$$D \leftarrow S \textbf{ op } D$$

式中的**op**常常称为*RasterOp*（光栅操作）或写模式，通常由16种布尔运算符组成。SRGP支持其中最常用的几种，包括置换（**replace**）、或（**or**）、异或（**xor**）和与（**and**）；在图2-16中以1位/像素的图像为例说明了四种操作的差异。

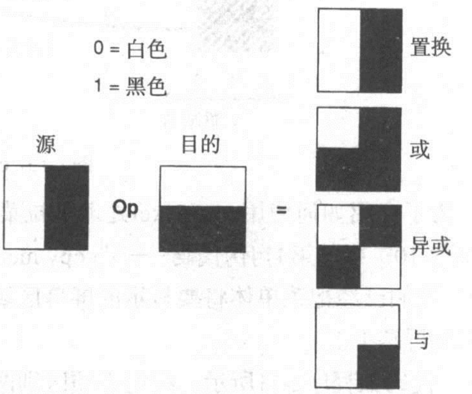


图2-16 结合源像素和目的像素的写模式

写模式影响的不仅是SRGP_copyPixel，还有所有写到画布上的新图元。每个像素（SRGP_copyPixel操作的源矩形像素或图元像素）存储于各自的内存单元，写操作时既可以采用破坏性的**replace**模式，也可以将它的值与当前存储的像素值进行逻辑结合。（这个源值与目的值的逐位结合类似于在读-修改-写内存循环中CPU硬件对内存单元的内容执行算术或逻辑操作。）尽管**replace**是最常用的模式，**xor**模式在生成动态对象（例如，我们即将要讨论的光标和橡皮回应）时也很有用。

可以调用以下函数设置写模式属性：

```
void SRGP_setWriteMode ( writeMode WRITE_REPLACE / WRITE_XOR /
    WRITE_OR / WRITE_AND );
```

由于所有的图元是根据当前的写模式生成的，SRGP程序员必须确定已显式地设置该模式，而

不能依赖于默认设置WRITE_REPLACE。

为了了解RasterOp的工作原理，我们现在来看软件包内部是如何完成像素的存储和操纵的。这是我们从抽象角度讨论光栅图形至今，惟一一次涉及硬件和具体实现的问题。

RasterOp是对作为颜色表索引的像素值进行操作，而不是对颜色表中存储的硬件颜色定义操作。这样，在一个二值的1位/像素的系统中，RasterOp的操作对象是两个1位的索引。在一个8位/像素的颜色系统中，RasterOp则对两个8位的索引做逐位逻辑运算操作。

尽管图2-16中显示的四个基本运算对于1位/像素单色图像的操作很直观，但对于n位/像素图像（$n>1$），除**replace**模式以外的其他模式的结果都很不自然。这是因为对源索引和目的索引的按位逻辑运算将产生第三个索引，而该索引的颜色值可能与源颜色和目的的颜色完全不同。 54～55

replace模式涉及改写屏幕（或画布）上已有的内容。这种破坏性写操作是绘制图元的正常模式，通常用来移动和弹出窗口。它也可用于以应用程序背景图案重绘的方式“擦除”旧图元。

二值显示中的**or**模式将待显示图像非破坏性地加到画布中的原图像上。颜色0代表白色背景，颜色1代表黑色背景，如果在白色背景上以一个灰色填充图案做或（**or**）运算，会改变原来的位并显示出灰色图案。但如果在黑色区域上以一个灰色图案做或（**or**）运算，将不会对屏幕产生影响。因此，如果在一个以砖块图案填充的多边形上以一道亮灰色的画线做或（**or**）运算，将只以画刷的图案填充砖块，而并不会像**replace**模式那样擦除砖块的边界。所以，在绘图中常用**or**模式（见习题2.6）。

二值显示中的**xor**模式可以用来反转一个目的区域。例如，为了突出显示用户选择的按钮，我们设置**xor**模式并用颜色1生成一个填充矩形图元，这样就转换了该按钮的所有像素：0 **xor** 1 = 1，1 **xor** 1 = 0。为了恢复按钮的原来的状态，我们只要保持**xor**模式并再次画矩形，这样就把这些位转换到原来的状态。SRGP内部也利用这个技术来提供定位器的橡皮线和橡皮矩形回应模式（见例2.1）。

在许多二值图形显示中，**xor**技术被硬件（或在某些情况下的软件）用来以非破坏性的方式显示定位器的光标图像。这种简单的技术也有它的缺点：当光标位于几乎50%黑色和50%白色的细致背景图案上时，光标可能无法看清。因此，许多二值显示器和大多数彩色显示器使用**replace**模式回应光标并且使用带白色边界的黑色光标，以便它可以在任何背景上显示；不能做**xor**使软件和硬件回应机制更加复杂（见习题2.4）。

and模式可以用于有选择地将目的区域内的像素复位为颜色0，进而“清除”它们。

例2.1

问题：不采用内置定位器回应，实现一个橡皮筋回应。特别在交互循环的起点和终点上监视其中间结果。

解答：我们将实现一个橡皮筋框的回应。橡皮筋线回应能被类似地实现。当鼠标正在移动时，定位器的最后值尚未选定，交互功能将跟踪鼠标，并画出橡皮筋回应。在用户已经按下鼠标的按钮后，交互循环将被调用。交互将允许用合适的按钮来拖动当前点，直到用户释放按钮、指向交互的终点为止。

当我们的函数返回时，我们需要将状态重新存储到函数的入口，以便使调用者与我们的函数的实现细节相隔离。

创建回应的基本思路是使用xor方式来画出回应形状（矩形），这样可以通过重画同一形状来擦除它，而不必专门关注该回应叠加的是什么内容。为使回应图像出现，我们必须画它一次。为再现回应前的显示器状态，我们必须精确地重画一次同一形状，以免在屏幕上留下一些回应的踪迹。 56

由于我们必须收集定位器的数据来更新回应，因此在第一次采样定位前，需画出橡皮筋回

应，然后在一个循环里重复：采样，擦除旧的回应，再画更新后的回应。在擦除及再画前采样时，我们需检验按钮的状态，看看交互是否应终止。另外，检查一下最后采样时鼠标是否已经确切移动了，如果已移动，则只需进行擦除及再画即可，虽然这并不是基本的。

下面的C代码实现了该问题的上述解法（函数buttons_are_down()并没有展示，因为它只是简单地检验鼠标按钮的状态）：

程序2-12 实现橡皮筋回应交互的代码

```
void RubberRectInteract ( point anchor_pt, point curr_pt, int drag_flag,
        int buttonmask, locatorMeasure *final_loc, rectangle *final_rect)
{
    attributeGroup save_attributes;
    locatorMeasure curr_loc;
    int some_button_down;
    rectangle curr_rect;

    SRGP_inquireAttributes( &save_attributes );
    SRGP_setLineStyle( CONTINUOUS );
    SRGP_setFillStyle( SOLID );
    SRGP_setInputMode( LOCATOR, SAMPLE );
    SRGP_setWriteMode( WRITE_XOR );

    SRGP_setLocatorEchoType( CURSOR );              /* 或者NO_ECHO */
    SRGP_setLocatorMeasure( curr_pt );              /* 设置好度量 */

/*
 * 我们希望矩形对它的位置是否处在
 * 当前点的下面或左面不敏感——功
 * 能GEOM_实用程序，就是保证这点
 */
    curr_rect = GEOM_rectFromDiagPoints( anchor_pt, curr_pt );
/* 现在我们使橡皮筋框第一次出现： */
    SRGP_rectangle( curr_rect );

    while( (buttons_are_down( buttonmask, curr_loc.button_chord )) ) {
       SRGP_sampleLocator( &curr_loc );
/* 我们仅在鼠标器移动后更新回应 */
          if (curr_loc.position.x != curr_pt.x || curr_loc.position.y != curr_pt.y) {
             SRGP_rectangle( curr_rect );
/* 在此时，我们画的上一矩形框将不出现 */
             curr_pt = curr_loc.position;
             curr_rect = GEOM_rectFromDiagPoints( anchor_pt, curr_pt );
             SRGP_rectangle( curr_rect );
/* 现在矩形框将从新的位置出现 */
          }
    }

/* 在此时，我们在上一处已经画了一次矩形框 */
/* 因此我们必须通过再画一次而擦除它： */
    SRGP_rectangle( curr_rect );
    *final_loc = curr_loc;
    *final_rect = curr_rect;

/* 现在将状态重新存储作为功能的入口： */
    SRGP_setInputMode( LOCATOR, INACTIVE );
    SRGP_setAttributes( save_attributes );
}
```

57

2.4 SRGP的局限性

尽管SRGP是个能支持多种应用的功能强大的软件包，但其固有的限制使它对某些应用来说并不是最佳的选择。最显著的是，SRGP不能提供对3D几何显示的支持。另外，还有更多的甚至会影响许多2D应用的细微局限性：

- SRGP采用机器相关的整数坐标系，不适用于那些需要使用具有更高的精度、更大的范围、更加便利的浮点数的应用程序。
- 像其他大多数二维光栅图形库一样，SRGP以自由语义方式将图像以不连接的像素值矩阵形式存储在画布中，而不是以图形对象（图元）集合的形式，因此不支持对象级的操作，例如“删除”、“移动”、“改变颜色”。由于SRGP不记录生成当前屏幕图像的操作，一旦图像遭其他软件的破坏，它也无法刷新屏幕，另外它也不能通过重新扫描转换图元而将图像显示在具有不同分辨率的设备上。

2.4.1 应用程序坐标系统

在第1章中，我们介绍了下面的观点：对大多数应用程序，绘图仅仅是到达目标的一种手段，应用程序数据库的主要任务是支持诸如分析、模拟、验证和制造等过程。因此数据库必须按这些过程所要求的范围和精度存储几何信息，这与显示设备的坐标系和分辨率无关。例如，VLSI CAD/CAM程序中表示的线路只有1~2厘米长，需要精确到半微米，而一个天文学程序可 58 能要表示1~10^9光年的范围，精度一百万英里。为了达到最大的适应性和表示范围，许多应用程序在数据库中使用浮点数*世界坐标系*（world coordinate）存储几何特性。

这样的应用程序可以自己将世界坐标映射到设备坐标；但是，考虑映射过程的复杂性（我们将在第6章中讨论），更方便的做法是使用一个图形软件包，它接受以世界坐标表示的图元并以机器无关的方式把它们映射到显示设备中。如今市场上的廉价浮点芯片能提供与整数运算性能大致相当的浮点运算，这大大地减少了使用浮点运算所付出的时间代价——这样的花费对于适应性要求很高的应用程序来说是完全值得的。

对2D图形，提供浮点坐标的最常用的软件是Adobe公司的PostScript，它既是驱动硬拷贝打印机的标准页面描述语言，也是一些工作站上视窗系统的图形软件包（扩充版本Display PostScript）。对3D浮点图形，PHIGS和PHIGS + 已经得到普遍应用，PostScript中也出现了许多3D扩充功能。

2.4.2 为了重新定义存储图元

现在我们考虑如何使用SRGP以不同的尺寸重新绘制一幅图，或在具有不同分辨率的显示设备上（如高分辨率打印机）绘制大小相同的图。由于SRGP对已绘制的图元没有记录，应用程序必须在换算坐标后，为SRGP重新指定整个图元集合。

如果SRGP增强了保留所有指定图元记录的功能，应用程序就可以让SRGP通过读取存储器而重新生成图元信息。这样，SRGP便可支持刷新屏幕这个常用操作。在一些图形系统中，应用程序的屏幕图像会被其他用户或应用程序发出的消息破坏。在这种情况下，除非屏幕画布可以在屏外画布保存的冗余拷贝中刷新，否则修复被损坏的图元的惟一途径只有重新定义图元。

让软件包存储图元的最主要的优点是可以支持编辑操作，而该操作是制图或设计应用程序的核心。这些程序与我们在本章的例子中所描述的绘图应用程序很不相同。一个**绘图程序**（painting program）中用户可以使用可变尺寸、形状、颜色和图案的画刷绘制任意的线条。在更完善的绘图程序中用户还可以放置预定义的形状，如矩形、多边形和圆。画布的任何部分都能进行像素级编辑；对象的一部分可以被图画覆盖，画布的任意矩形区域也可以被拷贝或移动到别的地方。但是用户不能把一个已有的形状或画好的长条作为一个连续的不可见的对象进行删除或移动操作。这一限制的存在，是因为绘图程序允许画布上已有的对象被损坏或分割，这样就丧失了对象的连续性。例如，如果一个对象已被分割成散落在屏幕上不同区域的小片，那么 59

用户指向其中一个对象碎片意味着什么呢？用户是指碎片本身还是指原来的整个对象？从本质上说，具备单个像素操作能力的应用程序是无法实现关联拾取以及对象的拾取和编辑的。

相反，一个**制图程序**（drawing program）允许用户在任何时候拾取和编辑对象。这些应用程序也被称为**布局编辑程序**（layout editor）或**图形演示程序**（graphical illustrator），它们允许用户放置标准形状（也称为符号、模板或对象），并通过这些形状的删除、移动、旋转和缩放操作来编辑布局。类似地，允许用户用简单的3D对象组合复杂对象的交互程序称为**几何编辑器**（geometric editor）或**构造程序**（construction program）。

缩放、屏幕刷新和对象级编辑都需要应用程序或图形软件包对图元进行存储和重新定义。如果应用程序存储了图元，它就能完成重新定义；但是，这些操作远比它们看上去复杂得多。例如，图元的删除可以通过擦除屏幕并重新指定所有的图元（当然除了被删除的图元）完成；但是，更有效的方法是以应用屏幕背景重新绘制该图元，然后重新定义可能被破坏的图元。这些操作很复杂，但要经常使用，所以有必要让图形软件包来完成它们的功能。

对象级的几何图形软件包（如PHIGS）可以让应用程序使用一个2D或3D浮点坐标系统定义对象。软件包在内部将对象存储，应用程序可以对存储的对象进行编辑，并在编辑操作需要的时候随时更新屏幕。软件包也执行关联拾取，对给定的屏幕坐标生成相应的对象ID。由于这些软件包操纵的是对象，它们不允许像素级操纵（如copyPixel和写模式）——这就是保持对象连续性的代价。因此，没有图元存储的光栅图形软件包和有图元存储的几何图形软件包都不能满足所有的需求。第7章将讨论在图形软件包中保存图元的利与弊。

通过像素复制进行图像缩放

如果应用程序和软件包都没有图元的记录（这是大多数绘图程序的特征），缩放不能通过重新定义变换后的图元端点坐标来完成。只有通过读像素和写像素操作来缩放画布的内容。放大位图/像素图图像（使它变大）的一种简单、快速的方法是通过**像素复制**。这种方法用N乘N的像素块代表每个像素，即得到了放大N倍的图像。

像素经过复制后，图像变大，但由于没有提供新的除原始像素级以外的信息，图像也变粗糙了。另外，像素复制只能将图像的尺寸放大整数倍。我们必须使用另一种技术——区域采样和过滤（在第3章讨论），来实现正确的放大或缩小。过滤技术在深度大于1的像素图上能达到最好的效果。 60

图像缩放是经常会遇到的问题，尤其在要打印绘图程序创建的图像的时候。假设现在将画布传到一个分辨率是屏幕分辨率两倍的打印机上。各像素变成其原有尺寸的一半；因此我们只能以同样数目的像素和一半的尺寸来显示原始图像，或是通过像素复制来产生同样大小但分辨率为打印机本身性能一半的图像。总之，大小和质量不能兼顾。只有重新定义技术才能保证缩放的质量。

小结

在本章中，我们讨论了一个简单而功能强大的光栅图形软件包SRGP。应用程序可以使用它绘制具有不同显示属性的2D图元，这些属性影响图元的外观。图像可以直接在屏幕画布上或在任意尺寸的屏外画布上绘制。通过设置裁剪矩形属性，绘图可被限制在画布的一个矩形区域内。除了标准2D形状的绘制，SRGP还支持画布内或画布间的矩形区域拷贝。拷贝和绘制操作受写模式属性的影响，该属性使得目的像素的当前值对其新值的确定起一定作用。

SRGP引入了逻辑输入设备的概念，它是物理输入设备的高层抽象。SRGP键盘设备将物理键盘抽象化，而定位器设备将鼠标、数据输入板或游戏杆等设备抽象化。逻辑设备可以在采样（轮询）模式或事件模式下工作。在事件模式下，用户动作触发的事件信息放入事件队列，供应用程序随时访问。在采样模式下，应用程序要持续地访问设备度量以检测发生的变化。

由于SRGP扫描将图元转换为其成员像素，而不存储它们的原始几何形状，因此SRGP惟一允许的编辑操作是通过绘制一个新图元或通过使用对像素块的copyPixel操作来更改单个像素。应用程序必须自行实现对象的操作，如移动、删除或缩放等，并在SRGP中重新定义更新的图像。

其他系统为图像提供了不同的功能集。例如，PostScript语言支持浮点图元和属性，包括更自由的曲线形状和裁剪功能。PHIGS子程序软件包提供了分层建模的、定义在3D浮点世界坐标系中的对象的操作功能。这些对象存储在一个可编辑的数据库中；在任何编辑操作之后，软件包将根据存储信息自动重新生成图像。

SRGP是个子程序软件包，而且现在许多开发人员则认为解释语言，例如Adobe的PostScript，可以提供更强大的功能和适应性。至于子程序软件包（整数或浮点数，保留或不保留图元的）和显示语言（例如不保留图元的PostScript）哪个能成为标准，大家意见不一。它们各自有其适用的应用领域，而且都将继续使用下去。 61

在第3章里，我们将看到SRGP如何通过扫描转换和裁剪进行绘图。在后面的几章中，我们先要对硬件进行简要的介绍，然后讨论图形变换和3D视图的数学原理，为PHIGS的学习做准备。

习题

2.1 SRGP在视窗环境中运行，但不允许应用程序利用多窗口的优越性：屏幕画布被映射到屏幕上的一个窗口，而其他画布都不可见。如何改进SRGP的设计和应用-程序员界面，从而允许应用程序利用视窗系统的优势？

2.2 仅当SRGP应用程序只使用颜色0和1时，它才能完全实现机器无关。设计一种增强SRGP的策略，使SRGP能够在必要时模拟颜色，这样应用程序可以在二值显示器上正常工作的同时充分利用颜色特性。讨论这样的策略会产生的问题和冲突。

2.3 生成一个由一些普通对象移动和缩放组成的动画序列。首先通过擦除屏幕后重新定义对象的方法生成各帧。然后尝试*双缓存*，使用一个屏外画布作为每一帧被拷贝到屏幕画布之前被绘制到的缓冲区。比较这两种方法的结果。另外，考虑SRGP_copyPixel的用法。它在什么条件下有利于动画的生成？

2.4 在不使用SRGP的内置光标回应的情况下，实现非破坏性的光标跟踪。使用位图或像素图图案存储光标图像，图案中的0代表透明区域。在二值显示环境中实现异或（**xor**）光标，并在二值或彩色显示环境中实现置换模式光标。为了测试跟踪，应该用SRGP定位器设备执行一个采样循环，并在包含前面所写信息的区域上移动光标。

2.5 考虑在画图应用程序中实现以下功能：用户可以画一条异或（**xor**）线，将画刷经过的区域的颜色反转。可以设置写模式然后执行程序2-5中的代码，实现起来似乎很容易。这样会带来什么样的复杂情况？提出解决方案。

2.6 一些画图应用程序提供一种“喷漆绘图”模式，这种模式以随机方式对画刷扫过的区域内的少数像素着色。每次画刷扫过一个区域，会对不同的像素着色，所以画刷经过的次数越多，区域内着色的密度就越大。在二值显示环境中实现一个喷漆绘图交互过程。（注意：普通的算法会产生条纹，或者不能实现密度的增加。你必须创建一个稀疏位图或图案库；有关创建自定义图案的信息，请见参考手册。）

2.7 在不使用SRGP的内置文本图元的情况下，为二值显示实现透明背景文本。可以使用一个屏外画布存储不同字符的位图形状，但支持的字符数不要超过6个——这不是字形设计的课程！（提示：要用两种不同的算法来处理颜色0和1。） 62

2.8 制图程序在执行删除操作后，可以通过以应用屏幕背景图案填充被删对象的形状来更新屏

幕。当然这会损坏屏幕上的其他对象。为什么简单地通过重新定义所有与被删对象区域相交的对象来修复损失是不够的？试讨论优化图形修复问题的解决方法。

2.9 实现一个过程，在带细边界的不透明的矩形里居中绘制文本。调用者可以定义文本的颜色、背景和边界、“按钮”的中心所在的屏幕位置、宽和高的最小/最大尺寸以及字体和字符串本身。如果字符串长度超过按钮的最大行显示长度，则在合适的位置（如空格处）将该串断开，并以多行显示。

2.10 在屏幕上实现一个定值逻辑输入设备，用户可以使用鼠标改变模拟水银柱的长度来指定温度。设备的属性应当包括度量的范围、初始度量、度量的粒度（如精确到华氏2度）以及温度计的屏幕图像的长度和位置。为了测试设备，可以使用交互过程模拟一个无限等待状态（waitEvent），其中只将定值器激活。

2.11 假设通过在输入模型中增加一个屏幕上的定值器（与习题2.10中描述的类似）并同时支持事件和采样模式来定制一个SRGP实现。如果将该实现安装到一台只有一个物理定位器设备的工作站上会出现什么样的问题？提出解决方案。

2.12 实现一个“圆角矩形”图元——四个角是90°的椭圆弧的矩形。应用程序可以控制椭圆弧的半径。该图元支持轮廓和填充方式。

程序设计项目

2.13 实现一个下拉菜单软件包，其高层设计见2.2.6节、2.3.1节、2.3.3节中的代码段。软件包通过从输入文件读入字符串来初始化菜单条和菜单体。程序可以通过关闭菜单条使标题消失，也可以激活一个菜单（以各菜单标题在菜单条上的水平位置作为参数）来使菜单出现。

2.14 通过实现禁止所选菜单项这一功能来增强习题2.13中的菜单软件包。被禁止的菜单项应当灰显。由于SRGP不支持用画笔风格来绘制文本，所以在二值显示中，必须在写模式中覆盖实体文本以达到这种效果。

2.15 通过在用户选取菜单体上的项目时突出显示定位器当前指向的项目来增强习题2.13中的
63 菜单软件包。

2.16 实现一个布局应用程序，该程序使用户可以在屏幕上的一个方形子域内放置图形对象。程序应支持椭圆、矩形和等边三角形。用户可通过点击屏幕按钮来选择一种对象类型或初始化一个操作（重画屏幕、将场景保存到文件中、从文件中恢复场景或者退出）。

2.17 在习题2.16的布局程序中增加对象编辑功能。用户应当能够对对象进行删除、移动和调整大小等操作。可以使用以下简单的关联拾取方法：在应用数据库中扫描对象，并选择第一个其矩形域范围覆盖定位器位置的对象。（这种简单的方法有个很明显的缺陷：很可能一个可见的对象无法被拾取！）注意通过突出显示当前选中的对象给用户反馈。

2.18 在习题2.16的布局应用程序中，实现覆盖优先权功能。用户应当能够压下/弹出一个对象（改变其优先权为最低/最高）。增强关联拾取来使用覆盖优先权解决冲突。试分析在关联拾取中使用优先权并提供压下/弹出功能，是如何使用户可以克服原关联拾取的不精确问题的。

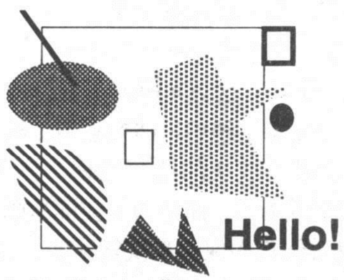
2.19 使用习题2.8的结果，优化习题2.16中的布局应用程序的屏幕更新算法，使响应编辑操作时需重新定义的对象数为最小。

2.20 增强习题2.16中的布局应用程序，以便同时启动键盘和定位器以支持常用操作的快捷键。例如，敲击“d”键可删除当前选择的对象。

2.21 为习题2.16的布局应用程序所支持的三种图形对象设计并实现关联拾取的分析技术。新的技
64 术应具有百分之百的准确性，而用户也不必再通过弹出/压下来拾取一个优先权低的可见对象。

第3章 二维图元的基本光栅图形学算法

光栅图形软件包近似做出数学意义上的（“理想的”）图元，这些图元定义在笛卡儿坐标系的网格点上，用适当亮度或色彩的像素点集来表示。这些像素一般作为位图或像素图存储在CPU内存或帧缓存中。在前一章，我们讨论了图形包SRGP的特点。从应用程序员的角度看，这是一个典型的光栅图形包。在本章，我们则从一个软件包实现者的角度来讨论SRGP，探讨将图元转换到像素的扫描转换算法，即如何根据图元的特点，在一个直立的矩形裁剪框内画出图元。图3-1中所示的是关于图元扫描转换及裁剪的一些例子。



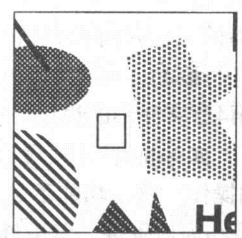


图3-1 在一个矩形裁剪框内裁剪SRGP的图元。a) 图元和裁剪框，b) 裁剪结果 65

在一些比较成熟和功能强的软件包中，采用了一些高级算法来处理一些SRGP并不支持的操作。本章讨论的算法是基于二维整数笛卡儿坐标网格的，而其中大多数扫描转换算法可推广到浮点数的情况，其裁剪算法则可推广到浮点数及三维的情况。本章最后一节讨论反走样的概念，即通过调节像素的亮度来尽可能地消除图元显示时的锯齿状表现。

3.1 概述

3.1.1 显示系统体系结构的含义

在第1章中介绍的基本概念模型中，我们给出了一个图形软件包。它介于应用程序（及其数据结构和模型）和显示硬件之间，以便为应用程序使用硬件提供一个与设备无关的接口。如图3-2所示，SRGP的程序可以分成两部分，一部分是输出流水线，另一部分是输入流水线。

在**输出流水线**中，应用程序根据应用模型或数据结构中存储的或推导的图元和属性对物体进行描述，并将这些信息传递给图形包，由图形包将它们裁剪和扫描转换为最终在屏幕上显示的像素。图形包中的图元生成程序确定要生成什么图元，其属性程序确定要怎样生成图元。SRGP_copyPixel程序确定对图像进行怎样的修改，而画布控制程序确定在什么地方生成图像。在**输入流水线**中，在显示终端的用户交互操作由图形包的采样程序或事件驱动输入程序转换成度量信息，并将这些度量信息传递给应用程序。然后，应用程序根据这些度量信息对模型或者屏幕上的图像进行修改。与输入相关的程序包括：初始化和控制输入设备的程序，以及在交互过程中从输入设备获取度量信息的程序。在本书我们将不讨论SRGP的画布管理和它的输入处理，因为这些内容主要是一些数据结构和底层的系统软件问题，与光栅图形学关系不大。

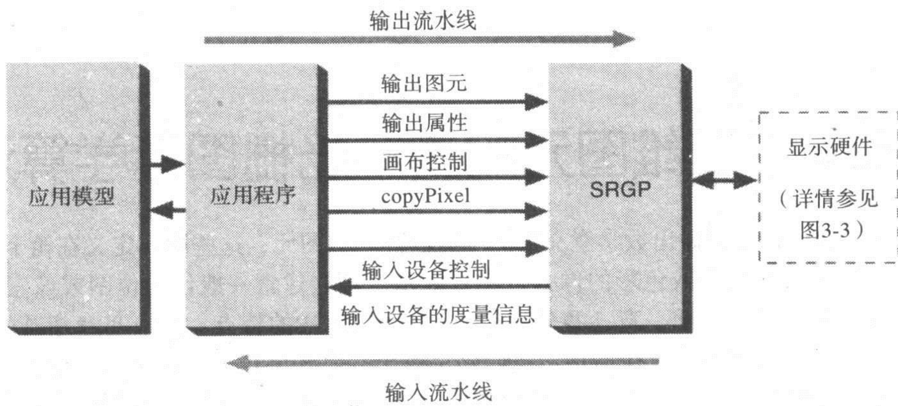


图3-2 作为应用程序和图形系统的中介，SRGP提供了输出流水线和输入流水线

实现一个SRGP图形包必须充分考虑各种显示设备。有些显示系统带有自己的帧缓存和显示控制器，这些显示控制器的工作是解释和执行绘图命令，将生成像素写入帧缓存。另有一些简单的系统只可直接由CPU进行刷新。其图形包的只输出部分可以驱动光栅硬拷贝设备。这些各具特点的硬件体系结构，在第4章有详细的讨论。在任何一个显示系统中，CPU必须能够对帧缓存中的每个像素进行读写操作。如果能够对帧缓存中的像素进行成块的读写操作，则能很方便地实现copyPixel(bitBlt)类型的操作。这一功能不是为了直接生成图元，而是为了使屏幕画面外的位图或像素图成为可见的，以及在窗口管理、菜单处理、滚动等操作中保留和恢复屏幕上的一些显示片段。

可直接由CPU进行刷新的系统的实现过程基本上是一致的，因为这些刷新工作都是由软件来完成的。然而显示控制器和硬拷贝的系统的实现过程则差别较大，这取决于硬件设备能做哪些工作以及哪些工作要由软件去完成。显然，在任何体系结构中，那些不能由硬件直接支持的图元和特征，必须通过软件进行扫描转换来生成。让我们简单地看一下体系结构及其实现的有关内容。

1. 具有帧缓存和显示控制器的显示器

如果SRGP驱动的显示控制器自己能够进行扫描转换并直接处理SRGP的图元和属性，那么SRGP只需做很少的工作。在这种情况下，SRGP只需将它关于图元、属性和写模式的内部表示转换成显示外设能够直接绘图的方式即可（参见图3-3a）。

如果存储映射允许CPU直接访问帧缓存，显示控制器也能直接访问CPU，那么显示控制器的体系结构的功能将是最强的。此时，CPU用其自身的指令就能对单个像素或copyPixel像素块进行读写操作，而显示控制器亦能在屏幕画面外的画布上进行扫描转换，并用它的copyPixel指令在两个存储器之间或它自己的帧缓存内进行像素的传递。当CPU和显示控制器可以异步运行时，则必须具有同步机制以避免存储器读写的冲突。通常，CPU是将显示控制器作为一个协处理器进行管理。如果显示设备的显示控制器只能在它自己的帧缓存中进行扫描转换，而不能将
66~67 像素写入CPU的内存，则我们要找出一种方法以便在屏幕画面外的画布上生成图元。在这种情况下，图形包可以用显示控制器在屏幕的画布上进行扫描转换，而对于屏幕画面外的画布，就必须用它自身的软件进行扫描转换。当然，图形包可以通过copyPixel操作将硬件数据扫描转换形成的帧缓存中的图像复制到屏幕画面外的画布。

2. 只有帧缓存的显示器

在没有显示控制器时，SRGP就必须自身进行扫描转换以生成屏幕画面外的画布和帧缓存图像。这种情况的一个典型结构如图3-3b所示，SRGP驱动一个存储共享的帧缓存。请注意，

图中只给出了那些作为帧缓存和存储由SRGP管理的画布的存储器。另有一些存储器是为一般软件和数据（包括SRGP自身）所用的，则未包括在内。

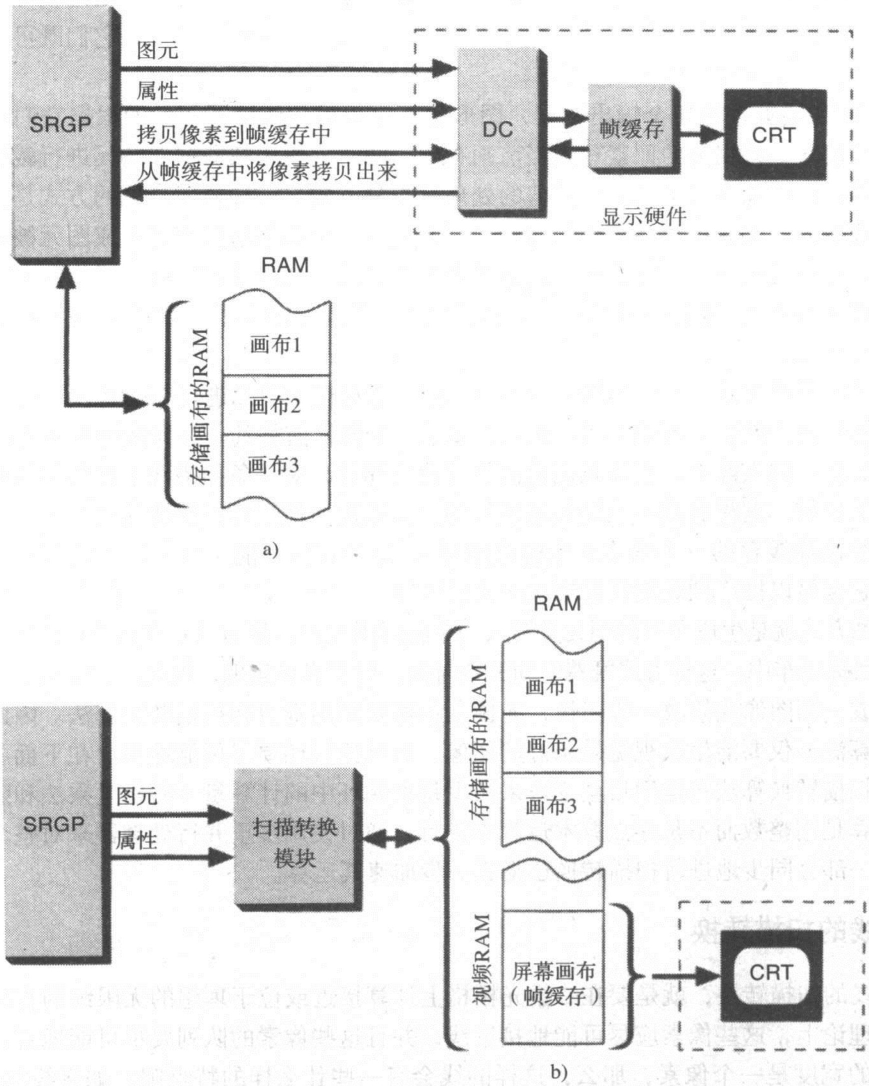


图3-3 SRGP驱动的两种显示系统。a) 带有显示控制器和帧缓存的显示外设，b) 无显示控制器，只有存储共享的帧缓存的显示系统

3. 硬拷贝设备

如同在第4章将介绍的那样，具有不同功能的硬拷贝设备多种多样。最简单的设备一次只接受一条扫描线，并且在胶片或纸上绘制时要靠软件来定位扫描线。对于这种简单的硬件，SRGP必须生成完整的位图或像素图，并一次扫描出一条线传递给输出设备。好一点的设备，一次可接受一整页（帧）。而功能更强的设备则自身带有扫描转换硬件，它们通常被称为光栅图像处理器（RIP）。作为最高档的外设，PostScript打印机有自己的内部“引擎”，以读取描述页面的PostScript程序（该描述与设备无关），这些“引擎”能解释这种程序以生成图元和属性，并随后进行扫描转换。由于基本的裁剪和扫描转换算法在本质上与光栅设备的输出技术无关，

因此，我们在本章对各种硬拷贝设备不再做过多的论述。

3.1.2 软件中的输出流水线

在此，我们只分析驱动简单帧缓存显示器的输出流水线，只是为了引出用软件进行裁剪和扫描转换的问题。将要介绍的各种算法在讨论时与具体设备无关。因此，它们既可以用于软件实现，也可以用于硬件或微代码实现。

对于SRGP要处理的每个输出图元，图形包都要*扫描转换*该图元，即根据它们可用的属性和当前的写模式，将相关的像素写入当前画布中。同时，要用裁剪框对图元进行**裁剪**，即图元中不在裁剪框内的像素将不显示。裁剪的处理有多种方式。一种显而易见的方式是在扫描转换以前进行裁剪，即先解析地计算图元与裁剪框的交点，再根据这些交点生成图元被裁剪后的结果。显然，这种处理方式的好处是：扫描转换操作只需要处理裁剪后的图元，而不是原来的图元（它可能大很多）。这种技术经常用来裁剪线条、矩形和多边形，因为处理它们的裁剪算法相当简单和有效。

最简单的裁剪方式，称为**截剪**（scissoring），即对整个图元进行扫描转换，但只显示位于画布上裁剪框内的像素。理论上讲，就是在显示一个像素前将其坐标与裁剪框边界的(x, y)坐标区间进行比较。但实际上，正如我们后面将讨论的那样，对一条扫描线上相邻的像素可以避免一些这样的比较。这种剪裁一般是快速进行的。如果边界检查可以快捷地进行（例如，利用微代码或指令高速缓存的一个紧凑工作的内循环），这种方法可能实际上比先裁剪再扫描转换的方法快。它也可以推广到处理任意形状的裁剪区域。

68~69

第三种方式就是生成所有的图元并写入一个临时画布中，然后只对在裁剪框内的像素进行复制以送到目标画布中。这种方式既费空间又费时间，但它容易实现，因此，它常用来处理文字。

每生成一幅图像或修改一幅图像，光栅显示都要调用裁剪和扫描转换算法。因此，裁剪和扫描转换算法不仅要能生成视觉效果好的图像，而且执行还要尽可能地快。在下面各节的详细讨论中，扫描转换算法将运用*增量方法*来减少每次循环中的计算量（特别是乘法和除法运算），并且其运算是用整数而不是浮点算术运算来进行。通过使用多重并行处理器来对整个输出图元或图元的一部分同步地进行扫描转换还将进一步加速其运算。

3.2 直线的扫描转换

对直线的扫描转换，就是要在二维光栅格上计算接近或位于理想的无限细的直线上的像素的坐标。理论上，这些像素应尽可能地接近线，并且这些像素的队列要尽可能地直。假设对线近似表达的宽度是一个像素，那么，这样的线会有一些什么样的特点呢？如果线的斜率在1和-1之间（包括1和-1），则每一列上必定只有一个像素被显示；若线的斜率在此范围之外，则每一行上必定只有一个像素被显示。无论长度和方向如何，整条线应该以相同的亮度尽可能快地生成。此外，还要为画线提供其他的功能：所画的线可以宽于一个像素，其宽度相对于理论上的线中心对称；它可以具有不同的线型和笔型，可以具有高质量图形所需的其他效果。例如，在程序员的控制下，端点区域的形状可以是斜的、圆的和勾角的。在每个像素具有多位的显示系统中，运用反走样技术可以加强像素的亮度变化能力，这样，我们就可以尽可能地改善对线进行离散的拟合所产生的锯齿形状。

至此，我们只讨论每列只显示一个像素的斜线的情况（若是陡的线，则每行上只有一个像素显示）。在本章后面的部分，我们将讨论宽的图元及其处理方式。

我们知道，为了可视化几何属性，SRGP将一个像素表达成一个圆形的点，并且其中心就

是像素在整数栅格上的(x, y)坐标。这种表达方式便于拟合CRT电子光束的近似圆形的横截面，但实际显示时，光束斑点之间的间距在不同的系统中变化很大。在有些系统中，相邻的斑点是相重叠的；而另有一些系统中，垂直方向相邻的像素之间不相重叠；在大多数系统中，水平方向上的间距要比垂直方向上的间距小。在不同的系统中，坐标系的表达也有差异，比如在苹果牌Macintosh机器中，像素位于相邻栅格线构成的矩形框内，而不是在栅格线上。这样，在数学上由两个对角点定义的矩形在这种机器上就由此矩形内的所有像素来表示。如此就存在零宽度的画布，比如，由(x, y)到(x, y)定义的矩形不包含任何像素。而在SRGP中，这个画布会有一个在该点的像素。至此，我们依然将像素表达为不相接的圆，它们的圆心位于栅格点上。只有在讨论反走样时，此定义才略有改变。 70

图3-4中所画的是放大了很多倍的一条单个像素宽的线和它要拟合的真实的线，实心圆表示被显示的像素，空心圆则表示没有被显示的像素。在真实的屏幕上，圆形像素的直径要大于像素间的间距，所以这种符号表示实际上是夸大了像素的离散性。

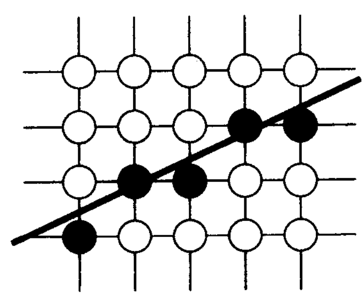


图3-4 实心圆表示扫描转换一条线所显示的像素

因为SRGP的图元是定义在整数栅格上的，所以线的端点是整数坐标。但实际上，若先用一个矩形框裁剪线段，则线段与裁剪边的交点作为裁剪后的端点，其坐标很可能是非整数的。对于浮点数光栅图形包，情况也一样。（在3.2.3节中，我们将讨论非整数交点的情况。）在下面的讨论中，我们将假设线的斜率$|m| \leq 1$。至于其他斜率的情况，只需做适当的调整就可以处理。而对于水平线、垂直线、斜率为± 1的这些常见的线，只需作为平常的特例处理即可，因为它们只可能穿过像素中心（见习题3.1）。

3.2.1 基本增量算法

对线的扫描转换，最简单的策略就是将斜率m计算为$\Delta y/\Delta x$，然后，从最左端的点开始，对x每次递增一个单位，而对每个x_i，计算其相应的$y_i = mx_i + B$，并显示坐标为（x_i，$\text{Round}(y_i)$）的像素，其中，$\text{Round}(y_i) = \text{Floor}(0.5 + y_i)$（即对$0.5 + y_i$进行取整）。这种计算是为了选择最接近线的像素，即到实际的线距离最短的像素。当然，这种简单的方式并不很有效，因为每次循环都要用浮点（或二进制分数）计算一次乘法、一次加法并调用一次取整运算。我们可以去掉其中的乘法，由于

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

当$\Delta x = 1$时，$y_{i+1} = y_i + m$。

因此，x每增加一个单位，y就加上一个m，m是线的斜率。对于线上的所有点(x_i, y_i)，我们知道，如果$x_{i+1} = x_i + 1$，那么$y_{i+1} = y_i + m$；也就是说，x和y的值可以根据前一点的值推算出来（见图3-5）。这就是将该方法称为增量算法的原因：在每一步，我们只需根据前一步的结果进行增量计算即可。

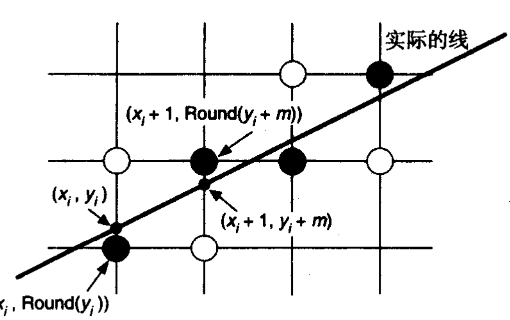


图3-5 关于(x_i, y_i)的增量计算

增量运算从一个端点的整数坐标 $(x_0$，

y_0)开始。值得注意的是，增量算法避免了对y轴上的截距B进行任何处理。如果$|m|>1$，当x变化一步时，y变化的步长将大于1。此时，我们就得将x和y的角色进行倒换，即y每次增加一个单位，x变化的增量为$\Delta x = \Delta y/m = 1/m$。程序3-1中的程序Line实现了这种增量运算。其起始点必须是左端点，并且只适于$-1 \leqslant m \leqslant 1$的情况。其他斜率的情况，只需要相应地做一些调整即可。在此我们省略了对水平线、垂直线或对角线等特殊情况进行检测的处理。

程序3-1 增量的线扫描转换算法

```
void Line(int x0, int y0, int x1, int y1, int value)
{                                  /* 设 - 1≤m≤1, x0 < x1 */
    int   x;                       /* x以单位步长从x0增长到x1 */
    float dy, dx, y, m;

    dy = y1 – y0;
    dx = x1 – x0;
    m = dy / dx;
    y = y0;
    for (x = x0; x <= x1; x++) {
        WritePixel(x, (int) floor(y + 0.5), value);    /* 置像素的值为value */
        y += m;                                         /* y移动的步长是斜率m */
    }
}
```

Line中的WritePixel是由设备层的软件提供的一个底层程序，它的作用是将一个值写入画布中的一个像素，该像素的坐标由它的前两个参数确定[⊖]。在此，我们假设只在替换（**replace**）模式下进行扫描转换。对于SRGP的其他写模式，我们必须用一个底层的ReadPixel程序去读取目标像素，并将该像素与源像素进行逻辑操作，然后用WritePixel将结果写入目标像素。

这一算法常被称为**数字微分分析器**（DDA）算法，DDA是用数值方法求解微分方程的一种机械设备，即根据x和y的一阶导数，在x和y方向上渐进同步地以小步长移动，由此生成连续的像素坐标(x, y)。在目前所考虑的情况下，x方向上的增量为1，y方向上的增量为$dy/dx = m$。由于计算机中实数变量的精度是有限的，不精确的m的重复迭加会产生累加误差并导致偏离实际的Round(y_i)值；然而对于大多数的（短）直线，将不会引起什么问题。71～72

3.2.2 中点线算法

上节中Line程序的缺点是：对y值取整要花费时间，而且，因为斜率是一个小数，y和m必须是实数或二进制小数。为此，Bresenham提出了一个只使用整数运算的经典算法[BRES65]，它能够根据前一个已计算的坐标(x_i, y_i)进行增量运算得到(x_{i+1}, y_{i+1})，而不必进行取整操作。该算法也可扩充其浮点数功能以处理端点坐标是任意实数的直线。Bresenham的增量技术还可以用来对圆进行整数的计算，尽管它还很难处理一般的二次曲线。因此，我们使用一种稍有不同的方法，叫作*中点技术*。该技术最早由Pitteway发表[PITT67]，而后Van Aken[VANA84]和另外一些研究人员对它进行了一些改进。Van Aken[VANA85]指出：中点方法在处理直线和整数型的圆时就简化成了Bresenham方法，会产生同样的像素。Bresenham证明并得出结论：通过对误差（离实际图元的距离）进行最小化处理，他的直线和圆的整数型算法能给出对精确图元的最好的逼近结果[BRES77]。在[KAPP85]中，Kappel讨论了多种误差标准的结果。

在下面的讨论中，我们将假设直线的斜率在0和1之间。对于其他斜率的情况，根据主轴对称的方式作适当调整就能处理。我们还假设左下端点为(x_0, y_0)，右上端点为(x_1, y_1)。

⊖ 如果不存在这样的底层程序，可以使用SRGP_pointCoord程序，请见SRGP的参考手册。

考察图3-6中的线，黑色的圆表示已被选择的像素，当前被选中的像素后面的两个空心圆表示下一步要选择的候选像素。假设我们刚选择了在(x_p, y_p)的像素P，下一步要选择的像素可能是其右边的第一个像素E（称为东像素），也可能是其右上的第一个像素NE（称为东北像素）。假设Q是要被扫描转换的线与栅格线$x = x_p + 1$的交点，根据Bresenham的方法，先计算E和NE到Q的垂直距离，然后根据这两个距离的差的符号挑选离Q最近的像素，作为被扫描转换线的最好逼近。用中点方法时，我们是考察中点M在线的哪一边。显然，如果中点在线的上方，像素E就更靠近线；否则，中点在线的下方，则是像素NE更靠近线。当然，线也可能在E和NE之间的中点穿过或两个像素都在线的同一边。但不管哪种情况，中点检测的方法都会选择最靠近的像素。此外，这样处理的误差（即被选择的像素到实际线的垂直距离）一定小于等于1/2。

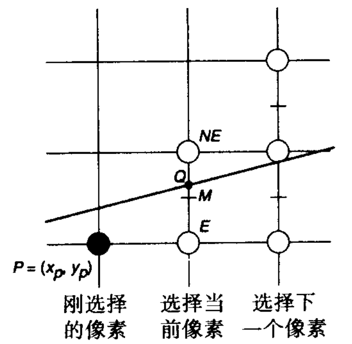


图3-6 中点线算法的像素栅格，包括中点M和两个候选像素E和NE

在图3-6中，被选择的像素是NE。现在，我们讨论如何计算中点在线的哪一边。假设直线 73
由其隐式函数⊖ $F(x, y) = ax + by + c = 0$表示。（在斜截式中，$y$的系数$b$与$y$方向上的截距$B$无关。）如果$dy = y_1 - y_0$, $dx = x_1 - x_0$，其斜截式可以写为

$$y = \frac{dy}{dx}x + B$$

因此，

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

其中$a = dy, b = -dx, c = B \cdot dx$⊜ 。

容易证明，对于线上的点，$F(x, y)$等于0；对于在线下方的点，$F(x, y)$是正数；而对于在线上方的点，$F(x, y)$是负数。运用中点法则时，我们只需计算$F(M) = F(x_p + 1, y_p + 1/2)$并考察它是正数还是负数。由于是根据在点$(x_p + 1, y_p + 1/2)$的函数值进行判定，我们定义一个**判定变量**$d = F(x_p + 1, y_p + 1/2)$。根据定义，$d = a(x_p + 1) + b(y_p + 1/2) + c$。对于$d>0$和$d<0$，我们分别选$NE$和$E$。如果$d = 0$，两者都可以选，我们在此就选$E$。

当栅格线移到下一条时，我们考察中点M的位置和d的值是如何变化的。当然，它们都依赖于我们是选择了E还是NE。如果选择了E，M就沿x方向递增一步。那么

$$d_{new} = F(x_P + 2, y_P + \tfrac{1}{2}) = a(x_P + 2) + b(y_P + \tfrac{1}{2}) + c$$

但是，

$$d_{old} = a(x_P + 1) + b(y_P + \tfrac{1}{2}) + c$$

从d_{new}中减去d_{old}得到一个增量差，因此，$d_{new} = d_{old} + a$。

我们将选择E后所得的增量称为Δ_E，$\Delta_E = a = dy$。换句话说，不必直接计算$F(M)$，我们就能根据当前的判定变量值得到下一步判定变量值，即简单地加上Δ_E。

如果选择了NE，M在x和y方向上都要移动一步。那么，

$$d_{new} = F(x_P + 2, y_P + \tfrac{3}{2}) = a(x_P + 2) + b(y_P + \tfrac{3}{2}) + c$$

⊖ 这个函数形式可以很方便地扩展成关于圆和椭圆的隐式方程。

⊜ 为确保中点算法的正确操作，选择a为正数是很重要的；在此，由于$y_1>y_0$，如果dy是正数，该条件可得到满足。

从d_{new}中减去d_{old}得到一个增量差，我们将其写为

$$d_{new} = d_{old} + a + b$$

我们将选择NE后所得的递增量称为Δ_{NE}，$\Delta_{NE} = a + b = dy - dx$。

74 基于上面的讨论，增量的中点技术可以概括为：在每一步，我们可根据上一步所得的判定变量值的符号去选择下一个像素；然后根据所选择的像素，用Δ_E或Δ_{NE}递增判定变量的值。

因为第一个像素就是第一个端点(x_0, y_0)，我们能直接计算d的初始值，由此来选择E或NE。第一个中点的坐标是$(x_0 + 1, y_0 + 1/2)$，因此

$$\begin{aligned} F(x_0 + 1, y_0 + \tfrac{1}{2}) &= a(x_0 + 1) + b(y_0 + \tfrac{1}{2}) + c \\ &= ax_0 + by_0 + c + a + b/2 \\ &= F(x_0, y_0) + a + b/2 \end{aligned}$$

由于(x_0, y_0)是在线上的点，所以$F(x_0, y_0)$等于0；因此，d_{start}就是$a + b/2 = dy - dx/2$。根据d_{start}我们可得到第二个像素，然后依此类推得到后续的像素。为去掉d_{start}中的小数部分，我们对原来的F略加修改，即将它乘以2变成$F(x, y) = 2(ax + by + c)$。这样，方程中的常量和判定变量都被乘以2。但这不会影响判定变量值的符号，因此也不会影响中点检测的效果。

我们已经看到，计算d_{new}时只需用简单的加法而无需费时的乘法。特别是，如程序3-2中的中点算法所示，算法中的内循环非常简单。在这个循环中，首先是检测d以选择一个像素，但实际上，我们是在修改了判定变量后再对当前像素在x和y方向上递增（这是为了与绘制圆和椭圆的算法一致）。注意，程序3-2中所示的算法只适应于斜率在0和1之间的线，对于一般情况的处理被留作为习题 3.2。在[SPRO82]中，通过对原先基本的算法进行一系列的程序变换，Sproull很巧妙地推导出这一算法的Bresenham方法。而对于圆和椭圆算法却尚未见到类似的推导，但其后正如我们将会看到的，中点技术事实上作了推广。

程序3-2 中点线扫描转换算法

```
void  MidpointLine(int x0, int y0, int x1, int y1, int value)
{
   int   dx, dy, incrE, incrNE, d, x, y;

   dx = x1 – x0;
   dy = y1 – y0;
   d = dy * 2 – dx;                  /* d的初始值 */
   incrE = dy * 2;                   /* 选择E时所用的增量 */
   incrNE = (dy – dx) * 2;              /* 选择NE时所用的增量 */
   x = x0;
   y = y0;
   WritePixel(x, y, value);             /* 起始像素 */
   while (x < x1) {
      if (d <= 0) {                     /* 选择E */
         d += incrE;
         x++;
      } else {                          /* 选择NE */
         d += incrNE;
         x++;
         y++;
      }
      WritePixel(x, y, value);          /* 所选择的像素最靠近线 */
   }
}
```

75

如图3-7所示，画一条从点(5, 8)到点(9, 11)的线，d值的连续变化是：2, 0, 6和4，其像素的选择顺序相应地为NE, E, NE和NE。为了清晰地揭示算法的几何特性，像素的间距和所画的像素均人为地放大了，这使得线的锯齿状特别明显。基于同样的原因，在下面各章节图中所画的图元都要比它们在真实屏幕上所显示的要粗糙。

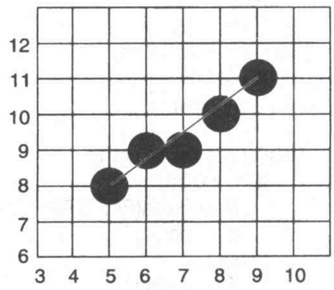
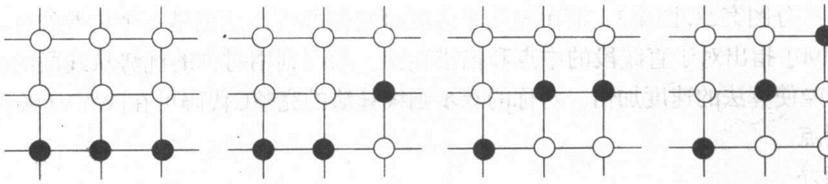


图3-7 从点 (5, 8) 到点 (9, 11) 的中点线

例3.1

问题：开发并编写一个中点线算法的增强版本的程序，要求该算法预测两个像素而不是一个像素。

解答：采用扫描转换的中点线算法的一个可选方案，是由Wu和Rokne [WU 87]介绍的双步算法。和中点算法相像，该技术也是增量的；采用整数算术由 (x_i, y_i) 计算出 (x_{i+1}, y_{i+1})。中点算法首先确定线段的方向（斜率），再继续在每一步中，通过一个判定变量在两个可选像素中选一。双步算法减少判定的次数，例如一次选中下一像素的两倍，即寻找下一对像素而不只是一个像素，这样继续进行下去。Wu [WU87]给出了四种可能出现的图案：



图案1　　图案2　　图案3　　图案4

Wu证明在同一线段上，不会同时出现图案1 和图案4 。此外，如线段斜率大于1/2，图案1不会出现；类似地，如线段斜率小于1/2，图案4不会出现。因此，通过测试斜率，选择将局限于三种图案之一：1, 2, 3或2, 3, 4。让我们看一下当斜率在0到1/2之间的情形，像已经指出的那样，它已排除了图案4。为用于填充光栅图像，需要在图案1、图案2、图案3中做出决定。判定变量d置初值$d = 4dy - dx$ 。于是，每一次增量（每步两个光栅单元）测试是否$d < 0$以确定是否使用图案1。如果d大于或等于0，则必须在图案2和图案3之间做出决定。这是一个简单的测试：$d < 2dy$。我们使用下列规则，对d予以增量：

$$d_{i+1} = d_i + 4dy \qquad d_i < 0\ （图案1）$$

$$d_{i+1} = d_i + 4dy - 2dx \qquad 其他（图案2或图案3）$$ 76

利用这一改进的优点，中点线算法能修改成实现增强的中点线算法的代码：

```
void DoubleStep( int x0, int y0, int x1, int y1 )
{
    int current_x, incr_1, incr_2, cond, dx, dy, d;

/* 内循环代码用于（0 < 斜率 < 1/2）情形。
DrawPixels例程需要图案和当前x位置两个参数。
这是因为上一像素可能只需画一个像素，而不是
整个图案。如画整个图案，线段可能扩展太快 */

/* 对内循环的设置初始值 */
    dx = x1 – x0;
```

```
    dy = y1 – y0;
    current_x = x0;
    incr_1 = 4*dy;
    incr_2 = 4*dy – 2*dx;
    cond = 2 * dy;
    d = 4*dy – dx;

    while( current_x < x1 ) {            /* 仍需要更多的扫描转换 */
      if ( d < 0 ) {                     /* 第一次判定 */
        DrawPixels(PATTERN_1,current_x);
        d += incr_1;
      } else {
        if ( d < cond )                  /* 不是第一种情形，是2，还是3 */
          DrawPixels(PATTERN_2, current_x);
        else
          DrawPixels(PATTERN_3, current_x);
          d += incr_2;
      }
      current_x += 2;
    }
}
```

像中点算法一样，所有计算均可使用整数加法和乘法（乘2）来进行。如果显示器是多级别的，那么可不区分图案2和图案3，简单地用半亮度画它们即可，这也是一个免费的反走样形式！Wyvill [WYVI90] 指出对于直线段的中点和扫描转换，均可利用对称的优势从线段的两个端点同时进行，这样能使算法的速度加倍。对称的双步画线算法的完整C代码可在[WYVI 90]中找到。

3.2.3 补充要点

1. 端点顺序

77 画线时必须考虑的一个问题是：画一条从P_0到P_1的线应该和画一条从P_1到P_0的线具有相同的像素序列，也就是说，画出的线应该与线的端点顺序无关。在前面的讨论中，像素选择有赖于线的方向的惟一情况是线正好穿过中点而判定变量是0的时候。此时，从左向右扫描时我们会选择E（东像素）。根据对称，如果是从右向左扫描，我们就要选择W（西像素），而这样所选择的像素，就要比从左向右扫描时所选择的相应的像素在y方向上高出一个单位。因此，在从右向左扫描时，当判定变量$d=0$时我们就应该选择SW（西南像素）。关于其他斜率的情况，需做类似的调整。

一般地，通过改变给定线的端点顺序，就能使扫描转换的操作总是按同一个方向进行。但在显示具有线型的线段时，就不能这样处理。线型操作应该总是从线的起始点开始使用设定的写掩码，而按前述方法（交换起始点），使得起始点总是位于左下端的点，与线的方向无关，这样处理可能会产生不正确的效果。例如，对于点划形式的线型，如111100，当我们希望这个线型从所定义的起始点开始操作而不是直接从左下点开始时，就可能出现问题。再者，如果算法以规范次序排列端点，线型在运作时可能对一个线段是从左往右进行，而对另一个邻接的线段则可能是从右往左进行，因为它们的斜率不同。这样，在这两条线段的共享点处就可能出现不希望的断裂情况，而实际上线型在相邻的线段间应该是无缝过渡的。

2. 起始于裁剪矩形的边的线

我们必须修改我们的算法以处理被裁剪算法裁剪掉一部分的线。（裁剪算法将在3.9中讨论。）图3-8a 中画了一条线，它被裁剪矩形的左边剪去一段，左边是：$x=x_{min}$。这条线与左边线的交点的x坐标是一个整数，而y坐标是一个实数。根据增量算法，在x_{min}值处为这条线所选

择的像素正是左边上的像素($x_{\min}$, Round($mx_{\min}+B$))⊖。给出了此初始像素值以后，我们还须初始化下一列上像素E和NE之间中点处的判定变量。在此，选择正确的像素序列是很重要的。否则，将这条线在边界$x_{\min}$处裁剪后，再用整数型的中点线算法对裁剪后从($x_{\min}$, Round($m\ x_{\min}+B$))到(x_1, y_1)的线进行扫描转换，就会产生偏差，因为线的斜率发生了变化。

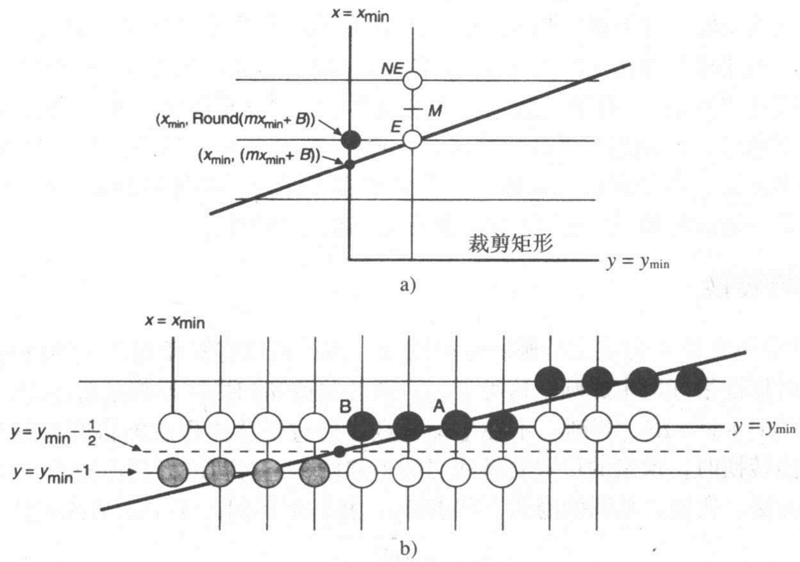


图3-8 从裁剪边界开始画线。a) 与一条垂直边相交，b) 与一条水平边相交（灰色的像素表示它们在线上但不在裁剪矩形内）

如果线与裁剪矩形的水平边而不是垂直边相交，则情况要复杂一些，如图3-8b所示。对于所示的线为窄线类型时，会有多个像素在对应于裁剪区域的底边的扫描线$y=y_{\min}$上。这些像素应该都包含在裁剪矩形内。但如果只简单地计算这条线与扫描线$y=y_{\min}$的交点，再对交点的x坐标取整，就会得到像素A，而不是所示的几个像素中最左的B。从图中可清楚地看到，像素B是当线刚刚越过中点$y=y_{\min}-1/2$时所在位置的右上方向的像素。因此，我们只需求线与水平线$y=y_{\min}-1/2$的交点，再对交点的x坐标取整，就能得到第一个像素B的坐标(Round($x_{y_{\min}-1/2}$), $y_{\min}$)。

最后要指出的是，即使端点是在一个浮点数光栅图形包中定义的，增量中点算法亦同样适用，只是增量由一个浮点数表示，算术运算也按浮点数方式进行。

3. 根据斜率变化线的亮度

观察图3-9中所示的两条扫描转换线。对角线B的斜率是1，因此它的长度是水平线A的$\sqrt{2}$倍。而这两条线都是由相同数目的像素（10个）表示的。如果每个像素上的亮度是I，那么线A的单位长度上的亮度就是I，而线B上的就仅是$I/\sqrt{2}$。这种差别很容易被人们觉察出来。尚若显示器只有2种显示状态则无法消除这种差别。但在n位像素的显示系统中，我们可以根据斜率调整亮度以消除这种差别。在3.14节中我们将会看到，反走样技术能生成效果更好的线。它的处理是将线

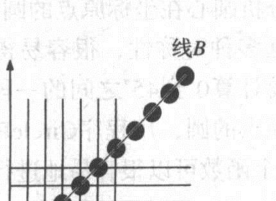


图3-9 光栅线的灰度值随斜率变化

⊖ 当$mx_{\min}+B$正好位于两条水平栅格线的正中时，我们实际上是向下取整，当$d=0$时，结果是选择像素E。

看成一个很薄的矩形，并对每一列上靠近或内含于矩形的多个像素计算出适当的亮度。

将线看成一个矩形也是生成宽线的一种方法。在3.7节中，我们将讨论如何修改基本的扫描转换算法以处理宽的图元以及具有不同线型和笔型等属性的图元。

4. 由线构成的轮廓图元

知道如何扫描转换线之后，我们如何扫描转换由线构成的图元呢？对于折线的情形，可以
78~79 每次扫描转换一条线段。对于矩形和多边形这些定义区域的图元，在扫描转换时可以一次扫描转换一条线段，但这样会画出一些不在图元所定义区域内的像素（参见3.4节和3.5节对这问题进行处理的一些特殊算法）。必须注意，对连接边共享的顶点只能画一次，因为对一个顶点画两次可能会改变色彩，比如当写屏幕的模式是**xor**时，就会变成背景色；若写到胶片上，亮度值就会加倍。实际上，两条靠近或交错的线会共享多个像素。参见习题3.7中关于这个问题的讨论，以及关于一条折线和一列连接起来的线段之间的差异的讨论。

3.3 圆的扫描转换

尽管SRGP中不支持圆但它支持椭圆这种图元，因此可以将圆弧当作一种特殊的椭圆弧并根据圆的八重对称性很好地生成圆。这种处理对圆的裁剪和扫描转换都是适用的。圆心在坐标原点的圆方程为$x^2 + y^2 = R^2$。若圆心不在坐标原点，通过平移操作就能将圆心移到坐标原点，但在其后的扫描转换时，像素的位置要做适当的平移修正。对圆进行扫描转换有几种容易实现但效率不高的方法。比如，从圆的隐式方程求解y，我们可得到$y = f(x)$的显式表达式为

$$y = \pm\sqrt{R^2 - x^2}$$

为了画出四分圆的一部分（其余四分圆根据对称性画出），我们可以将x一个单位一个单位地从0递增到R，并在每一步求出$+y$的值。但这样处理要进行乘法和二次开方的运算，其效率不高。特别是当x靠近R时，圆会有较大的间断，因为圆的斜率在此变得无穷大（见图3-10）。另有一种效率差不多的方法，就是将θ一步步地从0°增长到90°，并画点$(R\cos\theta, R\sin\theta)$；该方法能避免出现大的间断。

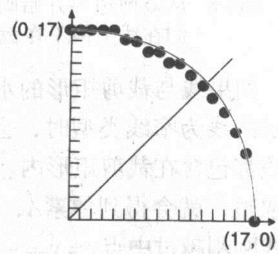


图3-10 x按单位步长变化生成的四分圆，y根据x的值计算并取整。由于相应于每个x值只生成一个y值，因而产生了间断

3.3.1 八方向对称性

利用圆的对称性，我们可以改进上节所介绍的画圆方法。首先分析圆心在坐标原点的圆。如图3-11所示，若点(x, y)在圆上，则根据多种对称性，很容易得到另外 7 个在圆上的点。为此我们只需要计算0°到45°之间的一段圆弧就能得到整个圆。对于圆心在坐标原点的圆，用程序CirclePoints就能将8个各相对称的点显示出来（这个函数可以很容易地进行修改以处理圆心不在坐标原点的圆）：

80

```
void CirclePoints (float x, float y, int value)
{
    WritePixel (x, y, value);
    WritePixel (y, x, value);
    WritePixel (y, –x, value);
    WritePixel (x, –y, value);
    WritePixel (–x, –y, value);
    WritePixel (–y, –x, value);
    WritePixel (–y, x, value);
```

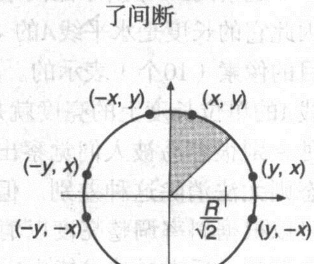


图3-11 圆上的8个各相对称点

```
    WritePixel (-x, y, value);
}
```

在$x = y$时，我们并不想调用CirclePoints程序，因为对于4个这样的点，每个都将被画两次。代码只需稍做修改以处理其边界情况。

3.3.2 中点圆算法

Bresenham[BRES77]开发了一种增量圆生成器，其效率要高于我们此前所讨论的方法。该算法假设用笔绘图仪的方式递增地沿着圆弧行进，能为圆心在坐标原点的圆产生所有在圆上的点。对于圆心坐标和半径均为整数的情况，我们运用中点准则推导了一个类似的算法，它能产生同样的一组优化的像素。特别是，此方法的程序代码本质上与专利4,371,933[BRES83]中所给出的一致。

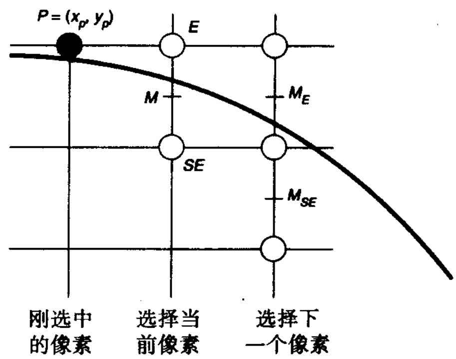


图3-12 中点圆算法的像素栅格，E和SE是两个候选像素，M是它们之间的中点 81

我们只考虑角度为45°的一段圆弧，即从$x = 0$到$x = y = R/\sqrt{2}$的第二个八分圆弧，并调用程序CirclePoints画出圆上其余的所有点。与中点线算法相似，这里所用的方法是：在2个像素之间的中点处给出一个评估函数值，并据此在2个像素中选择更靠近圆的那个像素。如图3-12中所示，如果在(x_p, y_p)的像素P是刚被选择的最靠近圆的一个像素，那么下一个像素就要在E和SE之间选择。

设函数$F(x, y) = x^2 + y^2 - R^2$。对于圆上的点，此函数的值是0；对于圆内的点，函数值是正的；而对于圆外的点，函数值则是负的。这样，如果像素E和SE之间的中点在圆外，则SE更靠近圆。相反，如果中点在圆内，则E更靠近圆。

与画线类似，我们选择像素是根据判定变量d的值，即函数$F(x, y)$在中点处的值：

$$d_{\text{old}} = F(x_P + 1, y_P - \tfrac{1}{2}) = (x_P + 1)^2 + (y_P - \tfrac{1}{2})^2 - R^2$$

如果$d_{\text{old}} < 0$，就选择像素E，并将当前中点的x坐标增加一个单位，以得到下一个中点，从而得到

$$d_{\text{new}} = F(x_P + 2, y_P - \tfrac{1}{2}) = (x_P + 2)^2 + (y_P - \tfrac{1}{2})^2 - R^2$$

由于$d_{\text{new}} = d_{\text{old}} + (2x_p + 3)$；因此，增量$\Delta_E = 2x_p + 3$。

如果$d_{\text{old}} \geqslant 0$，就选择像素$SE$[⊖]，而下一个中点是将当前中点的$x$坐标增加一个单位，$y$坐标减少一个单位，从而得到

$$d_{\text{new}} = F(x_P + 2, y_P - \tfrac{3}{2}) = (x_P + 2)^2 + (y_P - \tfrac{3}{2})^2 - R^2$$

由于$d_{\text{new}} = d_{\text{old}} + (2x_p - 2y_p + 5)$，可得增量$\Delta_{SE} = 2x_p - 2y_p + 5$。

回顾画线时，Δ_E和Δ_{SE}都是常量；但在处理圆这样的二次函数时，Δ_E和Δ_{SE}在每一步都要变化，且依赖于上一步所选择的像素P的坐标x_p和y_p。由于计算Δ_E和Δ_{SE}的公式是以(x_p, y_p)为自变量表达的，我们将P称为**估值点**。在每一步，我们直接将上一步所选择像素的x和y坐标值代入增量计算公式，就得到了新的增量。由于计算公式是线性的，其直接计算的开销并不大。

⊖ 当$d = 0$时选择SE，这与我们在画线算法中的选择是不同的，当然如此选择是随意的。读者如果以手工的方式模拟这个算法的运行就会发现，在$R = 17$时，这样选择会改变一个像素。

总之，和画线时一样，此算法的每次循环都做同样的两步：第一是根据上一次循环时所计算的判定变量d值的符号选择一个像素；第二是根据所选择的像素计算增量Δ，并用它修改判定变量d。与画线算法惟一的不同之处是：该算法在修改d时要根据估值点计算一个线性函数。

至此，我们还需讨论的就是计算初始条件。由于限定该算法处理的圆半径是整数，并只画第2个八分圆弧，因此，圆的起始像素是$(0, R)$。由于下一个中点的位置是$(1, R-1/2)$，因此$F(1, R-1/2) = 1 + (R^2 - R + 1/4) - R^2 = 5/4 - R$。该算法的实现如程序3-3所示。显而易见，这个算法的结构与画线的算法很相似。

因为d的初始值有小数，所以这个算法必须做实数的算术运算。尽管只需做适当的修改，该算法就能处理圆心不在整数光栅格点的圆和半径不是整数的圆，但是我们还是希望有一个效率更高的只进行整数运算的算法。为此，我们可对这个算法做一些简单的编程变换，以避免小数的出现。

82 首先，我们定义一个新的判定变量$h = d - 1/4$，即将程序中的d替换为$h + 1/4$。这样在初始化时，$h = 1 - R$。而比较$d<0$变成了$h< - 1/4$。然而，由于h的初始值是一个整数，并且其相应的增量（Δ_E和Δ_{SE}）也是整数，因此我们可将这个比较改为$h<0$。这样算法就变成了关于h的只进行整数运算的形式。为保持与画线算法的一致性，我们将全部的h替换成d，从而成为一个完整的整数型算法，如程序3-4所示。

程序3-3　中点圆扫描转换算法

```
void  MidpointCircle(int radius, int value)
{
    int   x, y;
    float  d;

    x = 0;                    /* 初始化 */
    y = radius;
    d = 5.0 / 4 – radius;
    CirclePoints(x, y, value);
    while (y > x) {
        if (d < 0) {          /* 选择E */
            d += x * 2.0 + 3;
            x++;
        } else {              /* 选择SE */
            d += (x – y) * 2.0 + 5;
            x++;
            y– –;
        }
        CirclePoints(x, y, value);
    }
}
```

程序3-4　扫描转换圆的整数型中点算法

```
void  MidpointCircle(int radius, int value)
{
    int   x, y, d;

    x = 0;                    /* 初始化 */
    y = radius;
    d = 1 – radius;
    CirclePoints(x, y, value);
    while (y > x) {
        if (d < 0) {          /* 选择E */
            d += x * 2 + 3;
            x++;
        } else {              /* 选择SE */
            d += (x – y) * 2 + 5;
            x++;
            y– –;
        }
        CirclePoints(x, y, value);
    }
}
```

83 图3-13显示了用这个算法产生的半径为17的圆的第2个八分圆弧，以及根据对称性产生的第1个八分圆弧（请与图3-10进行比较）。

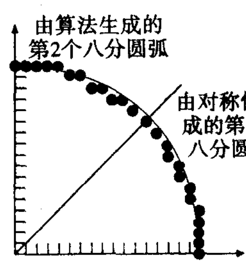


图3-13　运用中点算法生成的第2个八分圆弧，及由对称性生成的第1个八分圆弧

二阶差分

我们可以对增量计算方法进行进一步的拓展，由此改善画圆的中点算法的效率。我们知道Δ函数是线性方程，可以进行直接计算。然而任一个多项式都可以按增量的方式进行计算，如同我们处理线和圆的判定变量一样。事实上，我们是在计算**一阶和二阶的偏差分**，在第9章我

们还将使用这一有用技术。其根本思想就是计算函数在其两个邻近点上的值及这两个值的差分（对多项式而言，常常是一个低次多项式的值），并且在程序的每一次迭代中运用这个差分值。

在当前迭代中，如果我们选择了E，则估值点从(x_p, y_p)变化到(x_p+1, y_p)。显然，在点(x_p, y_p)处的一阶差分$\Delta_{Eold}=2x_p+3$。因此，在点(x_p+1, y_p)的$\Delta_{Enew}=2(x_p+1)+3$，且其二阶差分是$\Delta_{Enew}-\Delta_{Eold}=2$。

类似地，在点(x_p, y_p)处的$\Delta_{SEold}=2x_p-2y_p+5$，在点$(x_p+1, y_p)$处，$\Delta_{SEnew}=2(x_p+1)-2y_p+5$，因此二阶差分是$\Delta_{SEnew}-\Delta_{SEold}=2$。

如果在当前的循环中我们选择了SE，则估值点从(x_p, y_p)变化到(x_p+1, y_p-1)。因此在点(x_p+1, y_p-1)处$\Delta_{Enew}=2(x_p+1)+3$，而二阶差分是$\Delta_{Enew}-\Delta_{Eold}=2$。

同样，在点(x_p+1, y_p-1)处$\Delta_{SEnew}=2(x_p+1)-2(y_p-1)+5$，而相应的二阶差分是$\Delta_{SEnew}-\Delta_{SEold}=4$。

于是，修改后的算法包括以下几个步骤：(1)根据上一次循环所计算的判定变量d的值的符号选择一个像素；(2)运用上一次循环所计算的相关的Δ，用Δ_E或Δ_{SE}值对判定变量d进行修改；(3)根据像素的移动情况，用前面计算的差分常量修改Δ；(4)移动像素。Δ_E和Δ_{SE}使用起始像素$(0, R)$进行初始化。这样修改后的程序如程序3-5所示。

程序3-5 运用二阶差分的中点圆扫描转换算法

```
void MidpointCircle(int radius, int value)
{
    /* 这个程序运用二阶偏差分来计算增量 */
    /* 假设圆心在坐标原点 */
    int  x, y, d, deltaE, deltaSE;
         x = 0;                          /* 初始化 */
         y = radius;
         d = 1 – radius;
         deltaE = 3;
         deltaSE = 5 – radius * 2;
         CirclePoints(x, y, value);
         while (y > x) {
            if (d < 0) {                 /* 选择E */
               d += deltaE;
               deltaE += 2;
               deltaSE += 2;
               x++;
            } else {                     /* 选择SE */
               d += deltaSE;
               deltaE += 2;
               deltaSE += 4;
               x++;
               y––;
            }
            CirclePoints(x, y, value);
         }
}
```

3.4 填充矩形

填充图元的工作可以分成两部分：判定哪些像素要被填充（这取决于连续裁剪后图元的形状），决定用什么值来填充这些像素。我们首先讨论只用一种颜色来填充未被裁剪的图元，而在3.6节中，我们将讨论图案填充。一般而言，决定填充哪些像素包括下面的操作：用连续的扫描线对图元求交，并对由相邻的像素构成的位于图元内的跨段进行从左至右的填充。

在用一种颜色填充矩形时，对每一条扫描线上从矩形的左边到右边的所有像素，我们都给它们置成同样的值，也就是说，填充从x_{min}到x_{max}的每个跨段。跨段反映了图元的**空间相关性**，即对于一个跨段内的所有像素，或者从一条扫描线到另一条扫描线，图元的特征常常是不变的。为尽可能地利用相关性，我们一般只找那些会引起相关性变化的像素。对于只用一种颜色的图元，一个跨段内的像素都置成同样的值，这就有了**跨段相关性**。而只用一种颜色的矩形，则具有很强的**扫描线相关性**，即与矩形相交的那些连续的扫描线都是完全等同的。以后我们还会为一般的多边形运用**边相关性**。我们利用各种相关性，不只是为了扫描转换二维图元，还可以用来绘制三维图元，这将在13.1节中讨论。 84~85

对一个跨段内的多个像素进行相同的处理的能力是非常重要的，因为这样就可以往帧缓存中一次写一个字，以尽可能地减少费时的访问存储器的操作。对于一个二值显示器来说，我们每次写16个或32个像素。当然，如果跨段不是按字长对齐的，算法必须屏蔽掉字中那些不对应

像素的位。在此我们需要特别考虑写存储器的效率，这如同实现copyPixel命令一样，我们将在3.13节中进行简要的讨论。不过在我们的代码中，我们将主要讨论如何确定跨段，而不讨论写进存储器时的效率问题，参见第4章以及习题3.10。

为此，矩形的扫描转换就只是一个简单的嵌套**for**循环：

```
for ( y from ymin to ymax of the rectangle ) {  /* 扫描线 */
   for ( x from xmin to xmax ) {                 /* 跨段内的像素 */
      WritePixel(x, y, value);
   }
}
```

在这种简单直接的处理中会出现一个有趣的问题，此问题类似于扫描转换一个由具有共享像素的折线段构成的线集。当两个矩形有一条共享边时，如果我们依次扫描转换每个矩形，那么，共享边上的像素将被写两次。正如前面所讨论的，这不是所希望的。该问题反映了区域定义图元的一个更广泛的问题，即需要确定哪些像素是属于图元的而哪些像素不是。显然，对于这些图元而言，在数学上位于图元内部的像素将肯定属于该图元。但是，在图元边界上的那些像素呢？如果我们只考虑一个矩形（或只是从数学上考虑此问题），边界上的像素将包含在图元内。但因为我们要避免对共享边扫描转换两次，我们就必须确定一些规则，以便惟一地确定边界元素。

一个简单的规则就是：由一条边确定的包含图元的半平面如果位于该边的左方或下方，那么，这条边上的边界像素（即边上的像素）就不属于该图元。因此，左端的边和底端的边上的像素将会画出，但右端的边和上端的边上的像素将不画出。所以，共享的垂直边将“属于”有共享边的两个矩形中靠右的那个。这样，矩形中的一个跨段就是在左边封闭而在右边开放的一个区间。

对此规则有一些情况需要说明。第一，它可用于任意形状的多边形，而不只是矩形。第二，矩形的左下顶点还是会被画两次（对这种特殊的情况，我们还需其他规则来处理，这将在下一节中讨论）。第三，这个规则也可以用于非填充的矩形和多边形。第四，这个规则会导致每个跨段遗失其最右边的像素，以及每个矩形失去其最上端的一行。这些问题表明，对于避免在共享边（或潜在的共享边）上的像素画两次的问题，没有“完美”的解决方法。但相比于像素不显示或在**xor**模式下将像素置成不想要的色彩，人们一般还是认为省去右边和顶边上的像素要好些（视觉上少一些杂乱）。 86

3.5 填充多边形

下面要讨论的多边形扫描转换算法处理的多边形包括凸的和凹的，甚至是自相交或有内部空洞的。它一般通过计算位于多边形的左端边和右端边之间的跨段来实现。跨段的端点用增量算法计算，此算法从前一条扫描线与边的交点计算下一条扫描线与边的交点。图3-14中说明了多边形扫描转换算法的基本过程，其中包括一个多边形和一条穿过它的扫描线。扫描线8与边FA和CD的交点是在整数坐标上，而与边EF和DE的交点就不是。图中的交点，从a到d，都用小竖线标志。

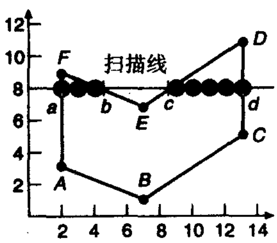


图3-14 多边形和扫描线8

我们必须决定每一条扫描线上的哪些像素在多边形中，并给这些像素置成适当的颜色（此图中的两个跨段分别是从x等于2到4和从x等于9到13）。为每一条与多边形相交的扫描线重复上述过程，就完成了对整个多边形的扫描转换，图3-15给出了使用这一操作的另一个多边形。

在图3-15a中，每个跨段的端点像素用黑色表示，而其间的像素用灰色表示。一种直接计

算跨段端点的方法是对每条边运用中点线扫描转换算法，并用一个表记录每条扫描线上的各跨段的端点。当为一条边产生的像素超越了已知的跨段时，就修改记录表。但是，这样会产生一些位于多边形外的端点像素，因为根据扫描转换算法，所选择的像素是最靠近边的，而没有考虑它们是否在多边形的内部——画线的算法没有内部与外部的概念。显然，对于有共享边的多边形，我们并不想画这些共享边以外的像素，因为它们侵入了相邻多边形的区域，当这些多边形是用不同的颜色填充时，这会使图形很难看。所以，最好是只画那些严格地在多边形内部的像素，即便一个外部像素可能更靠近多边形的边。为此，我们必须适当调整多边形扫描转换算法。比较图3-15a和图3-15b我们会发现，在图3-15b中有许多在理想图元外的像素没有画。 87

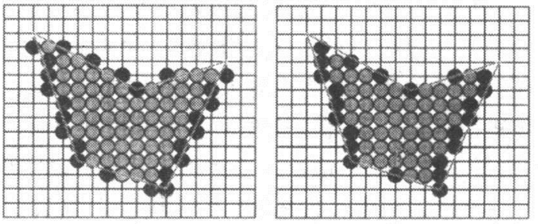


图3-15 一个多边形中的跨段。跨段的端点用黑色表示，跨段内的像素用灰色表示。a) 端点是用中点算法计算的，b) 端点位于多边形内部

使用这种技术，一个多边形就不会侵入别的图元所定义的区域（就是一个像素也不会）。我们可以运用相同的方法来处理非填充的多边形，以保证其一致性。甚至于当多边形没有相同的边界像素时，不管它们是填充的或非填充的，我们都可以用这样的技术在一条线段上同时扫描转换多个矩形和多边形。

同初始的中点算法一样，我们运用增量算法来计算一条扫描线上的跨段的端点，即根据前一条扫描线上的端点进行递推运算，而不是解析地计算扫描线与多边形的每条边的交点。例如，在图3-14中的扫描线8上，有两个在多边形内部的像素跨段。运用下面的3步操作，就能将这些跨段都填上：

1) 找到扫描线与多边形所有边相交的交点。

2) 根据x坐标的增序对这些交点进行排序。

3) 运用奇偶规则，填充每对交点间在多边形内部的像素。决定一个点是否在区域内的奇偶规则是：开始时，奇偶性初始化为偶，每当遇到一个交点就变换一次奇偶性；当奇偶性为奇时，就画像素，而当奇偶性为偶时，就不画像素。

在第1步和第2步是找到交点并对它们排序，这将在下一节讨论。现在，我们讨论如何填充跨段。在图3-14中，已排序的x坐标是（2, 4.5, 8.5, 13）。第3步需要考虑以下4个方面：

3.1) 如果交点是一个任意的有小数的x值，如何判定交点的哪一边的像素是在多边形内部？

3.2) 如果交点是在整数像素坐标上，如何处理这种特殊情况？

3.3) 若3.2中的交点是共享顶点，如何进行处理？

3.4) 在3.2的情况中，若顶点定义了一条水平边，该如何处理？

为了处理3.1的情况，如果以往右的方式靠近有小数坐标的交点，并且我们在内部，那么，我们对交点的x坐标进行往下取整的操作，以得到相关的在多边形内部的像素；否则，如果我们在外部，我们就往上取整，以保证所确定的像素在多边形内部。对于3.2的情况，我们运用的准则就是我们用来避免在矩形的共享边处出现冲突的准则：如果一个跨段最左的像素有一个整数的x坐标，我们就认为它在内部；如果其最右的像素有一个整数的x坐标，我们就认为它在外部。对于3.3的情况，在进行奇偶性的计算时，我们只对一条边上y_{min}顶点计数，而不对其有y_{max}顶点计数；这样，一个y_{max}顶点，只有当它恰是相邻边的y_{min}顶点时，才有可能画上。例如，图3-14中 88

的顶点A，在奇偶性的计算中，它只计数一次，因为它是边FA的y_{min}顶点，当然它也是边AB的y_{max}顶点。这样，边和跨段就都被当成了在最小值处关闭而在最大值处开放的区间。当然，相反的规则也是可行的。但这里使用的规则似乎更自然一些，因为它将最小值的端点作为进入点，而将最大值的点作为离开点。处理3.4的情况，也就是水平边的情况时，可以希望得到的结果是，像矩形一样，底部边被画上而顶部边则不画。正如我们在下一节将讨论的那样，如果不对这种（水平）边的顶点记数，上述结果将能自动实现，因为这些顶点既不是y_{min}也不是y_{max}顶点。

在图3-14中，扫描线8不与多边形的任何顶点相交。我们将上述的规则运用于该扫描线时，将填充下面这些像素：从a点（即像素(2, 8)）到b点左边的第一个像素（即像素(4, 8)）之间的所有像素，以及c点右边的第一个像素（即像素(9, 8)）到d点左边的第一个像素（即像素(12, 8)）之间的所有像素。对于扫描线3，因为顶点A既是边FA的y_{min}顶点，也是边AB的y_{max}顶点，因此，它被记数一次。这使得奇偶性变为奇，因此我们就从这里开始填充一个跨段的像素，一直到该扫描线与边CB的交点的左边的第一个像素，然后，奇偶性又变为偶。至于扫描线1，它只与顶点B相交，并且该点是边AB和BC的y_{min}顶点，因此，B将被记数2次，而奇偶性还保持为偶。这个点就相当于一个没有长度的跨段——在该点进入，画该点，然后又在该点离开。虽然在这样的局部最低点会画一个像素，但在局部最高点不会画任何像素。比如扫描线9与顶点F的交点。在此，顶点F是边FA和EF的y_{max}顶点，不会引起奇偶性的变化，奇偶性保持为偶。

3.5.1 水平边

考察图3-16中有关水平边的各种情况，我们不必对水平边的顶点进行有关奇偶性的记数就能够恰当地处理水平边的情况。在底端边AB，顶点A是边JA的y_{min}顶点并会引起奇偶性变为奇，而边AB上的顶点不影响奇偶性的变化，因此，奇偶性为奇而跨段AB会画出。由于垂直边BC的y_{min}顶点是B，它会使得奇偶性变为偶，但边AB此时也不会影响奇偶性的变化，因此，该跨段到此结束。在点J，它是边IJ的y_{min}顶点，而不是边JA的y_{min}顶点，因此，奇偶性变为奇，并画一个跨段到边BC。由于C是边BC的y_{max}顶点，不会引起奇偶性的变化，因此，该跨段沿着边CD继续前进，一直到D点为止。因为D是边DE的y_{min}顶点，使得奇偶性变为偶，并终止跨段。在点I处，由于I是边IJ的y_{max}顶点，而HI是水平边，所以，奇偶性保持为偶，而顶端边IH不会画出。但是在点H，由于它是边GH的y_{min}顶点，奇偶性变为奇，并从H开始画一个跨段，一直到该扫描线与边EF的交点的左边的第一个像素为止。最后，由于点G和F都不是y_{min}顶点，因此顶部边FG不会被画上。

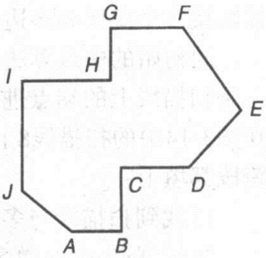


图3-16 一个多边形中的水平边

上面讨论的算法可以处理多边形中的共享顶点、两个相邻多边形的共享边以及水平边。它也可以处理自交的多边形。但是，我们注意到，该算法会遗失一些像素。更糟糕的是，如果不保持填充过程的记录，它会对一些共享像素画许多次，比如由两个以上的多边形共享的边，或者

89 两个不相邻的三角形所共享的y_{min}顶点（见习题3.11）。

3.5.2 狭长条

在我们的扫描转换算法中还有一个问题，它不能像水平边那样得到令人满意的处理，即多边形的边非常靠近，产生了狭长条——其多边形区域太过细窄以致于容纳不下扫描线上可见的最短跨段。例如，图3-17中的三角形，该三角形的三个顶点是(0, 0)，(3, 12)和(5, 12)。如果根据规则，只

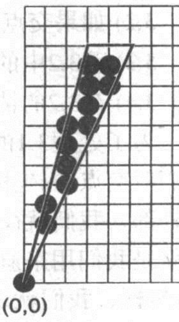


图3-17 扫描转换多边形的狭长条

画出在多边形内部以及多边形的左端边和底端边上的像素，许多扫描线上只有一个像素或根本就没有像素。这种“遗失”像素的问题，是走样问题的一个例子。以离散的方式对连续的信号进行逼近表达时，就会产生走样问题。如果像素是多位表示的，我们就可以应用反走样的技术，如应用在3.14节将介绍的关于线走样的处理方法。反走样将降低规则“只填充在多边形内部以及多边形的左端边和底端边上的像素”的作用，使边界像素甚至一些外部像素也能以某个亮度值进行填充。当然，这个亮度值是根据像素中心到图元的距离决定的。如此，在一个像素上可以汇聚多个图元的作用。

3.5.3 边相关性和扫描线算法

在算法的第一步，是计算扫描线与边的交点。这必须采用巧妙的方法，否则其算法运行速度将会很慢。特别是，我们必须避免采用直接了当的测试算法，即测试与每一条新扫描线相交的多边形的边。通常，对于给定的一条扫描线常常只有少数几条边是有价值的。而且，我们注意到，与扫描线i相交的许多边，也常常与扫描线$i+1$相交。对一条边而言，它与多少条扫描线相交，这种*边相关性*就会在这多条扫描线上存在。类似于中点线扫描转换算法，可以根据当前的点递推出下一个点，当从一条扫描线移到下一条扫描线时，我们可以根据上一条扫描线与边的交点的x坐标，很快地计算出新扫描线与该边交点的x坐标。其递推公式是：

$$x_{i+1} = x_i + 1/m$$

在此，m是边的斜率。在中点线扫描转换算法中，我们通过计算一个整型的判定变量避免了小数的算术运算，并且只需根据判定变量的符号就能选择最靠近实际线的像素。在此，我们也将用整型的算术运算进行相关的取整操作来计算最靠近边的在多边形内的像素。

现在，考察斜率大于$+1$的在左边的线。对于在右边或有其他斜率的线，可以用类似的参数进行处理，当然，对它们要做一些技巧性的处理。在此，注意垂直边是一种要特殊处理的情况（我们已知，水平边可以由跨段进行隐含的处理）。在线的端点(x_{min}, y_{min})，我们需要画一个像素。然后，当y按单位长度递增时，线上相关点的x坐标随之递增，其步长是$1/m$，在此，$m = (y_{max} - y_{min})/(x_{max} - x_{min})$是线的斜率。这样的增长使得$x$除了有一个整数部分外，还有一个小数部分，这可以表示为以$(y_{max} - y_{min})$为分母的一个分数。当我们重复这一过程时，小数部分将会溢出，而整数部分会增加。例如，如果斜率是$\frac{5}{2}$，并且x_{min}是3，那么x值的变化序列是 3, $3\frac{2}{5}$, $3\frac{4}{5}$, $3\frac{6}{5} = 4\frac{1}{5}$ 等等。当x的小数 90
部分是零时，我们就画上位于线上的像素(x, y)。当x的小数部分不是零时，我们需要进行取整操作来求得一个严格地在多边形内的一个像素。当x的小数部分变得大于1时，我们将x的整数部分加1，并将它的小数部分减1，同时将像素的位置往右移一位。如果增量运算到某处恰好落在一个像素上，就画出这个像素，并从x的小数部分中减去1，以保证小数部分小于1。

显然，当分子大于分母时，小数部分就大于1。因此，只需记录分数中分子的变化，就能避免使用小数。在程序3-6所示的算法中，我们实现了该技术。我们用一个变量*increment*来记录分子连续加时的变化。一旦分子大于分母，就从分子中减去分母，并将x坐标增加一个单位。

现在我们可设计**扫描线算法**，利用边相关性，对每条扫描线跟踪记录它所相交的边及其交点，存于一个称为**活动边表**（AET）的数据结构中。AET中的边是根据它们与扫描线交点的x坐标值进行排序的。这样，我们就能根据这些交点，按顺序依次两两成对地形成跨段进行填充，这些交点（经过取整处理）就是各个跨段的端点。当我们移到下一条扫描线$y+1$时，对AET要进行修改。首先，目前在AET中但不会与下一条扫描线相交的边（比如那些$y_{max} = y$的边）会从AET中去掉。然后，与下一条扫描线相交的新边（比如那些$y_{min} = y+1$的边）将被加入到AET

中。最后，对那些在AET中并且还将与下一条扫描线相交的边，我们将运用前面介绍的增量边算法计算新交点的x坐标。

程序3-6 扫描转换多边形的一条左边的边

```
void  LeftEdgeScan(int xmin,int ymin,int xmax,int ymax,int value)
{
    int   x, y, numerator, denominator, increment;

    x = xmin;
    numerator = xmax – xmin;
    denominator = ymax – ymin;
    increment = denominator;

    for (y = ymin; y < ymax; y++) {
        WritePixel(x, y, value);
        increment += numerator;
        if (increment > denominator) {
            /* 溢出，则向上取整以移到下一个像素，然后从分子中减去分母的值 */
            x += 1;
            increment –= denominator;
        }
    }
}
```

为了高效地将边加入到AET中，我们在初始化时建立一个全局的**边表**(ET)，它包含多边形的所有边，并且这些边按照它们各自y坐标较小值排序。
91 一般地，ET是基于桶排序的方式建立的，有多少条扫描线就有多少个桶。在每个桶中，根据边的较低的端点的x坐标，按照增序的方式排列各条边。ET中的每一项包含下列信息：边的y_{max}坐标值，边的下端端点的x坐标（x_{min}），随着扫描线递变到下一条线时的x坐标的增量$1/m$。图3-18中显示了图3-14中的多边形的6条边是如何排序的，而图3-19则显示了该多边形在扫描线9和10处的活动边表。（实际实现时，我们很可能会加一个标志，以区分是左端边还是右端边。）

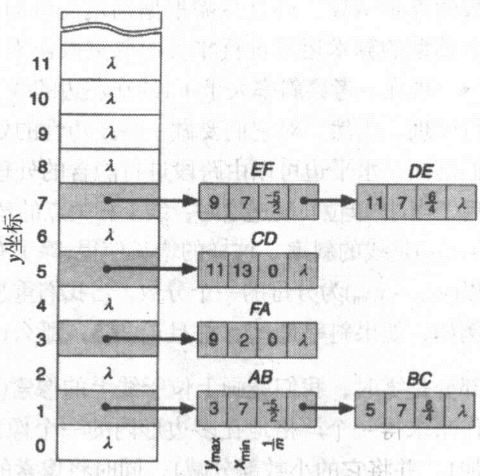


图3-18 为图3-14中的多边形建立的桶排序的边表

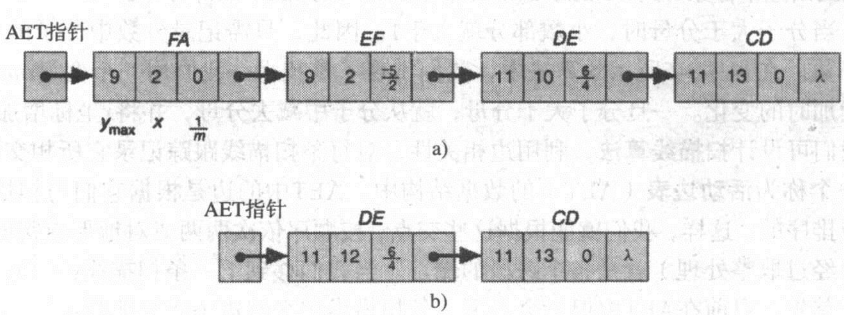


图3-19 为图3-14中的多边形建立的活动边表。a) 扫描线9，b) 扫描线10（注意：b)中边DE的x坐标是向上取整的，因为它是左端边。）

一旦生成了ET，扫描线算法就按下列步骤进行处理：

1) 将y置为边表ET中最小的y坐标值，即第一个非空的桶的y值。

2) 将AET初始化为空。

3) 重复以下运算，直至AET和ET都为空：

3.1) 从ET的y桶中选择那些$y_{min} = y$的边加入到AET中（进入边）。

3.2) 除去AET中那些$y = y_{max}$的项（即与下一条扫描线不再相交的边），然后在AET中根据x坐标值进行排序（这很容易实现，因为ET是已排好序的）。

3.3) 在扫描线y上，根据AET中的x坐标，两两成对地构成跨段，并对其中的像素以所希望的颜色进行填充。

3.4) y坐标增加1（即移到下一条扫描线的位置）。

3.5) 对于每一条留在AET中的非垂直的边，根据新的y值修改x坐标。

这个算法运用了边相关性来计算交点的x坐标，还利用了扫描线相关性（以及排序）来计算跨段。因为只对少量的边排序，并且在3.2步的重新排序操作是在一个差不多排好序或已排好序的链上进行，所以，无论是用插入排序的方法，还是用复杂度为$O(N)$的冒泡排序方法，在此都是可以接受的。在第13章和第14章中，我们将看到如何拓展这个算法用于处理多个多边形的可见性问题，包括处理透明多边形的情况。

在扫描转换图元的操作中，三角形和梯形可以作为特殊的多边形来处理，因为它们在每一条扫描线上只有两条边（水平边不进行显式的扫描转换）。实际上，因为任意一个多边形都可以剖分成有共享顶点和共享边的三角形网格（见习题3.14），因此对任意形状的多边形的扫描转换，我们都可以先将其多边形剖分成三角形网格，然后再扫描转换这些三角形。在计算几何中，这种三角剖分的处理是一个经典的问题[PREP85]，并且很容易应用于凸多边形；然而对于非凸的多边形，要进行有效的三角剖分是比较困难的。

注意跨段的计算是累积进行的。也就是说，在扫描转换算法的当前循环中运行到第3.5步，也就是在生成了同一条扫描线上的多个像素后，跨段的端点要进行适当的修改。（对于交叉边和狭长条的情况，在进行跨段的计算时要进行一些特殊的处理。）我们可以一次性地计算出所有的跨段，然后再对这些跨段进行填充，也可以边生成一个跨段就边填充一个跨段，直至处理完所有跨段。运用跨段的另一个好处是裁剪操作可以在计算跨段时同时进行：每个跨段可以独自地相对裁剪矩形的左边或右边的坐标进行裁剪。 92~93

3.6 图案填充

在前面几节，我们对SRGP中定义区域的图元进行填充时，只用一种颜色，其实现是将该颜色赋给WritePixel程序中的*value*参数。本节我们将讨论用图案进行填充，其实现方法是在扫描转换算法中最终写像素的时候增加一些控制操作。对于像素图图案，这种控制操作就是从图案中的适当位置选择出颜色，这将在下面讨论。当位图图案以透明方式填充时，我们执行WritePixel，对于图案中是1的位用前景颜色填充像素，而对于是0的位则不用WritePixel。这就像有关线型的处理一样。另一方面，如果位图图案以不透明方式填充，则对于1或0的位，就要分别选择前景颜色或背景颜色。

3.6.1 使用扫描转换的图案填充

用图案进行填充的最主要的问题是在图案区域和图元区域之间建立联系。换句话说，我们

要决定图案“固定”在哪里。这样，我们才能知道图案中的哪个像素相应于图元中的当前像素。

第一种方法是通过放置图案第一行最左的像素来将模式固定在多边形的一个顶点。这样，当图元移动时，图案可以跟着移动。对于具有较强几何结构的图案，比如在绘图应用中常用的交叉阴影线图案，该种处理可以产生理想的视觉效果。但是，一个多边形中并没有一个特别的点，可以很显然地建立这样的关联；至于圆和椭圆这样平滑变化的图元，就根本没有这样的点。因此，程序员必须确定建立这种联系的固定点是在图元的边界上还是在图元中。在有些系统中，一个固定点甚至会应用于一组图元。

第二种方法就是将整个屏幕用图案完全覆盖，而将图元当成是一个带透明性的轮廓或填充区域，以允许图案显示出来。在SRGP中使用了这种方法。这样处理，固定点的标准位置就是屏幕的原点。于是，图元的像素均被当成1，并同图案进行“与”的操作。这种方法的一个副作用是：当图元轻微移动时，图案不会“粘着”图元进行变化。相反，图元的移动就像是在固定的图案背景上进行剪切一样，因此，它的外观会随着其移动而变化。对于不带明显几何倾向性信息的规则图案，用户可能注意不到这种效果。除了计算效率高以外，这种绝对固定点的处理可以保证图元无缝地相互覆盖和邻接。

将图案应用于图元时，我们要根据当前像素的坐标(x, y)对图案进行索引以做对应的处理。因为一个图案是定义成一个$M \cdot N$的较小的位图或像素图，因此，我们需用“取模”的算术运
94 算来循环使用该图案。例如，如果将图案中的像素*pattern*[0, 0]与屏幕的原点重合[⊖]，我们就可以以透明的方式用下列的语句来写一个位图图案：

```
if ( pattern [x % M, y % N] )WritePixel(x, y, value)
```

如果我们以**replace**（替换）的写模式填充整个跨段，则当底层有一个copyPixel命令可以用于写多个像素时，我们就可以一次性地复制图案中的一整行。例如，假设图案是一个8 × 8矩阵，那么对于长度是8个像素的跨段，它就可以反复地进行这种操作。如果一个跨段的最左的点是字节对齐的（也就是这个像素的x坐标值对8取模后的值是0），那么，用copyPixel命令操作一个1 × 8的数组就可以将图案中整个第一行都写出。这种操作可以重复多次，直至填完跨段。如果跨段的两个端点都不是字节对齐的，则那些不在跨段内的像素必须用掩码屏蔽掉。程序员有时花费许多时间以使光栅算法在处理一些特殊情况时特别有效。例如，可以通过提前检测消除一些内循环，或者为内循环写一些汇编语言代码，以发挥某些专门硬件的长处，比如使用已介绍过的指令高速缓存或者一些特别有效的循环指令。

3.6.2 不用重复扫描转换的图案填充

至此，我们已经讨论了在扫描转换过程中进行填充的操作。另有一种方法，是先将图元扫描转换到一个矩形工作区，然后从该工作区的位图中将每个像素写到画布中的适当位置。这种写到画布的所谓*矩形写*的操作是一个简单的嵌套式**for**循环，此时，遇到1时，就用当前的色彩写；而遇到0时，就不写（透明时）或用背景颜色来写（不透明时）。与在扫描转换过程中进行填充相比，这种两步方法的工作量要多一倍。因此，对于只需被扫描转换一次的图元，这种方法是不合适的。但是，对于要被反复扫描转换多次的图元，这就比较合适。比如，给定了字体和大小的字符，就是这样的一种情况，可以提前对它们的轮廓进行扫描转换。对于只定义成位图字体的字符，或者别的什么作为位图作画或扫描的图元，如图标和一些应用符号，在任何情

⊖ 在窗口系统中，图案经常是固定在窗口坐标系的原点。

况下都不用扫描转换，而惟一要用的就是对位图实施矩形写操作。预先扫描转换位图的优点是，对一个矩形内的所有像素进行写的操作，不必做任何裁剪或有关跨段的算术运算，因此，与每次都从头开始进行扫描转换图元并做裁剪的操作相比，它显然要快得多。

既然我们要将一个矩形位图写到画布上，为什么不直接用copyPixel来复制位图，而是要一次只写一个像素？对于二值显示，用当前的颜色1进行写的时候，copyPixel可以很好地工作：对于透明的方式，我们用**or**（或）的写模式；对于不透明的方式，我们用**replace**（替换）的写模式。对于多值显示，我们不能用一个像素对应1位的方式直接写位图，而必须将位图中的每个位转换成一个完整的*n*位颜色值，然后再写。 95

一些系统有功能更强的copyPixel命令，它的拷贝操作可接受一个或多个关于“源读”或“目标写”的掩码的影响。如果我们可以将位图定义成一个“目标写”的掩码，将源定义成是（当前的）常量颜色的一个数组，我们就可以将这种功能用于透明的方式（用于SRGP中的字符处理）。那么，只有当位图的写掩码有1时，像素才被写以当前的颜色；此时，位图的写掩码就相当于一个具有任意形状的裁剪区。从某种意义上说，在一个*n*位像素的系统中以嵌套的**for**循环显式地实现矩形写的操作，就是模仿这种功能很强的“带有写掩码的copyPixel命令”工具。

现在考虑另外一种变形。我们希望画一个填充字母，或者别的什么形状，但是在其内部不是填实，而是用一种图案进行填充。例如，我们要用黑白均匀间杂的针点图案来填充一个粗体字符“P”，字形用一个50%灰度的图案，或者用一个具有两种色调的砖-灰泥图案来生成一个房屋的图标。那么，我们怎样才能用不透明的方式来写这样的一个形体而不进行反复的扫描转换呢？这个问题就是，位图中的0对应的区域内部的“洞”应当写以背景颜色，而区域外的洞（比如“P”中的空穴）还要以透明的方式写，以避免影响它后面的图像。换句话说，对于形体内部的0，我们要以背景颜色来标识，而对于其外部的0，包括空穴，就要根据写掩码保护形体外的像素。如果我们进行快速扫描转换，就不会出现位图中不同区域的0代表不同含义这种问题，因为，我们永远不会遇到形体边界外的像素。

我们可用一个四步过程来避免重复的扫描转换，在此，我们用图3-20a中所示的山的场景为例。运用图3-20b中图标的轮廓，第一步是生成一个将用作写掩码/裁剪区域的“填实”的位图，即图形内部的像素都设置为1，而其外部的像素都设置为0。这在图3-20c中已表达出来了，此时白色代表背景像素0，而黑色代表1。这种扫描转换只做一次。在第二步，一旦需要对形体进行图案化的拷贝时，我们将用背景颜色填实的位图以透明的方式写到画布上。这样，我们就以背景颜色清理出一块具有形体形状的区域，如图3-20d所示，在此，在已经有山的图像中，有一块房屋形状的区域被置成白色的背景颜色。第三步，用copyPixel命令，将一个矩形图案（图3-20e）按照**and**（与）模式作用于填实的位图，以生成其填实的形体的位图的图案化结果。这样，形体形状内部的

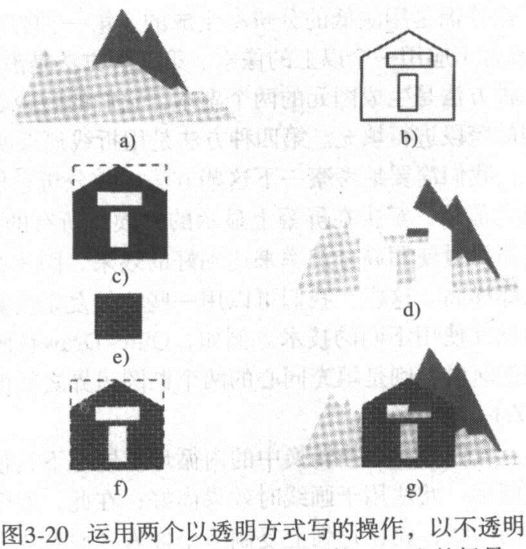


图3-20 运用两个以透明方式写的操作，以不透明的方式写一个图案化的形体。a) 山的场景，b) 房屋图标的轮廓，c) 房屋图标填实状态的位图，d) 通过写“背景”，清理出来的场景，e) 砖的图案，f) 砖的图案应用于房屋图标，g) 将图案化的房屋图标透明地写到屏幕上

一些像素就从1变到了0（图3-20f）。这可以看作是用形体的形状在任意大的图案中裁剪出一小块。最后，我们再将这个新的位图以透明的方式写到画布的同一位置。当然，这次是用当前颜色，即前景颜色，如图3-20g所示。这次与第一次写画布的时候一样，形体外部区域中的像素都是0，以保护区域外的像素，在此，区域内的0不影响以前写的（白色）背景；只有在有1的地方，才写（黑色）前景。如果想用一种带灰泥的红砖图案来对房子进行写的操作，我们可用灰色来写前面提到的填实位图，而用红色来写图案化的位图；在图案位图中，除了表示灰泥的一些狭带上是0外，大部分都是1。就效率而言，在此我们简化了矩形写的过程，该过程原本要在写掩码的作用下用两种颜色做图案填充，而现在则是在透明方式下做两次写操作或者带掩码的copyPixel操作。

3.7 宽图元

从概念上讲，仿照扫描转换单个像素宽的轮廓图元，我们就能生成宽图元。这就是将有某种横截面形状的画笔的中心（或者别的容易辨别的点，比如一个矩形画笔的左上角点）放在扫描转换算法所选择的像素上。单个像素宽的线可以看成是用大小只是单个像素的画笔所画出来的。当然，这样简单的描述掩盖了许多复杂的问题。首先，画笔的形状是怎样的？一般常用的画笔是圆形和矩形的。其次，非圆的画笔的朝向该是怎样的？矩形画笔是否总是直立的，以保持画笔固定的宽度？画笔的朝向是否能随着图元进行变化，使得画笔的垂直轴总是与图元相切？宽线的端点实际上该是什么样的形状？在整数栅格上又该是什么形状？一个宽多边形的顶
96~97 点会怎样变化？线型和笔型怎样相互作用？在本节，我们将讨论一些比较简单的问题，其余的问题在 [FOLE90] 的第19章中讨论。

画宽图元有4种基本的方法，如图3-21至图3-24所示。对于这些图元实际的情况，我们以白中夹黑的轮廓线表示；对图元进行单个像素宽的扫描转换所产生的像素，是用黑色表示的；为形成宽图元而增加的像素是用灰色表示的。旁边用黑色像素显示了缩小了的宽图元，其图像看上去好像是用颇低的分辨率生成的。第一种方法是粗糙的逼近，即在扫描转换过程中在每一列（或行）应用一个以上的像素。第二种方法是沿着图元单个像素宽的轮廓运动画笔的横截面。第三种方法是生成图元的两个副本，它们的间距是宽度t，然后对这两个副本所形成的内外边界之间的跨段进行填充。第四种方法是用折线逼近所有的图元，然后再用宽线画折线的每条线段。

我们简要地考察一下这些方法，并分析它们各自的优缺点。对于许多应用来说，即使不是绝大部分，至少在屏幕上显示的时候，所有的方法都能得到满意的效果。但是在用于印刷时，应当尽量使用高分辨率来达到好的效果，因为在印刷中，对于算法速度的要求不像实时生成图元那样高。这样，我们可以用一些比较复杂的算法来产生好看的结果。软件包甚至可能对不同的图元使用不同的技术。例如，QuickDraw在画线的时候是移动直立的矩形画笔，而在处理椭圆的时候，则是填充同心的两个椭圆边界之间的跨段。

3.7.1 复制像素

我们可将扫描转换中的内循环扩展一下，使它在每个计算出来的像素处写多个像素以生成宽图元。此法用于画线时效果尚好；在此，对于斜率在 - 1和1之间的线，就在每列上复制多个像素，而对于其他斜率的线就在每行上复制多个像素。当然，这样处理，线的端点总是垂直的或水平的，在画相当宽的线的时候，其效果是不大令人满意的，如图3-21所示。

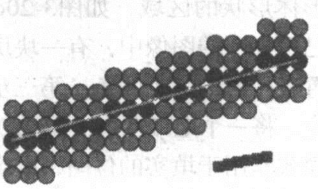


图3-21 通过列复制所画的宽线

复制像素的算法在几个线段以一个角度相接的地方还会产

生明显的缝隙，因为当斜率发生变化，需要从水平复制变到垂直复制，此时，这种算法会遗漏一些像素。对于椭圆，当其弧在45°的地方时，就会生成特别细的边界，如图3-22所示。

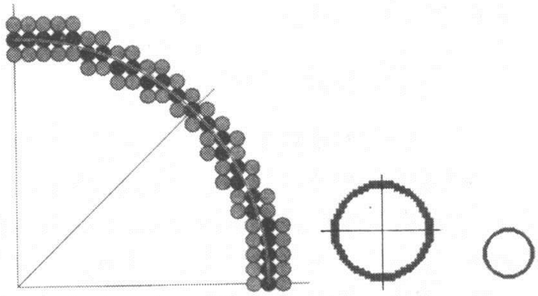


图3-22 通过列复制所画的宽圆

特别是，当图元的宽度定义为图元的内外边界之间垂直于图元切向的间距时，水平线和垂直线的宽度与倾斜的线的宽度是不同的。例如，如果宽度参数是t，水平线或垂直线的宽度是t，而倾斜角度是45°的线的平均宽度是$t/\sqrt{2}$。这又是一种倾斜线的像素比较少的情况，这种情况在3.2.3节中已经提过；与同样宽的水平线和垂直线相比，倾斜线的亮度就低。对于复制像素，还有一个普遍性的问题，即宽度为偶数时出现的情况。此时，我们无法将复制的列或行的中心置于所选择的像素上，这样，图元的一边比另一边就要“多出”一个像素。总之，像素复制的方法是一种有效但是略显粗糙的逼近方法，在画不是很宽的图元的时候，它的效果最好。

3.7.2 移动画笔

选择一个矩形画笔，使它的中心或角点沿着图元的单个像素宽的轮廓运动。对于线的生成，如图3-23所示，这样处理的效果相当好。注意，这条线与用像素复制的方法所生成的线很相似，但在端点处要宽一些。同像素复制的方法一样，因为笔是垂直对齐的，所画的图元的宽度是随着图元的角度变化的，但方式正相反：水平线段的宽度最细，而斜率为±1的线段最宽。例如，椭圆弧的宽度在它的整个轨迹上都是变化的，在其切线几乎是垂直或水平的弧段时，宽度正好是所定义的，而在切线的角度大约是±45°的时候，宽度变宽的因子是$\sqrt{2}$（见图3-24）。如果这个方块随着路径而转动，就可以解决这个问题。当然，最好是采用一个圆形的横截面，这样，宽度就与线倾斜的角度无关。

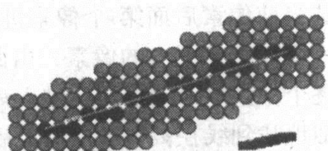


图3-23 通过跟踪一个矩形画笔所画的宽线

现在，对于一些简单的情形，如直立矩形或圆形的横截面，我们讨论如何实现这种移动画笔算法。最简单的方法是用命令copyPixel拷贝所要求的填实的或图案化的横截面（也称为*足迹*），以确保它的中心或角点就在所选择的像素上；对于一个圆形足迹或用不透明方式所画的图案，我们必须再用掩码屏蔽掉圆形区域外的一些位。然而这并不容易做到，除非底层的copyPixel有一个针对目标区域的写掩码。使用copyPixel完成任务的直接方式是对像素写多次，因为笔的足迹会在相邻的像素上重叠。一种比较好的方法是运用足迹的跨度来计算在相邻的像素上连续的足迹所形成的跨度，这方法也适宜于处理有关圆形横截面的问题。如同填充定义区域的图元，在一条光栅扫描线上跨段的组合不过是线段的合并，这只需要在每条光栅线上跟踪记录增长的跨段的最小和最大的x值。

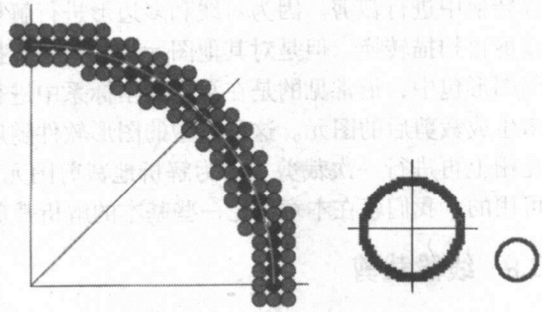


图3-24 通过跟踪一个矩形画笔所画的宽圆

显示宽图元的其他方法（如填充位于单像素轨迹两侧其距离为$t/2$的内、外边界之间的区域）

将在[FOLE90]的第3章和第19章中讨论。

3.8 光栅空间的裁剪操作

正如我们在本章概论中提到的那样，尽可能快地进行裁剪和扫描转换是非常重要的，以便在应用模型有所变化时能够很快地进行更新。裁剪可以解析地进行；在扫描转换过程中实时地进行；或者作为实际裁剪矩形的copyPixel命令的一部分，在将存储了未经裁剪的图元的画布复制到另一个画布上去的时候进行。上述第三种方法在以下情况下很有用：预先生成一个大的画布，然后用户根据需要移动裁剪矩形来检查这画布的一些部分，但不修改画布上的内容。

将裁剪和扫描转换结合起来，有时称为截剪，这在填充图元和画宽图元时容易做到，因为这可以作为跨段操作的一部分：只有端点需要裁剪，而内部点不需要检测。截剪也显示了跨段相关性的另一个优点。再者，如果一个轮廓图元并不比裁剪矩形大许多，即相对而言，图元落在裁剪区域外的像素不多。对于这样的情况，与提前进行解析的裁剪相比，在生成每个像素的同时实施裁剪的操作可能要快许多（即对其进行有条件限制的写操作）。特别是，尽管区间检测是在内循环中进行的，但大量的对外部像素的写存储器的操作可以节省，并且增量计算和检测操作可以完
98～100 全在一个高速存储器中（如CPU的指令高速缓存或者显示控制器的微码存储器中）进行。

另外，还有一些技巧可能是有用的。例如，在运用标准的中点扫描转换算法时，每次都挑选当前像素后面第i个像素进行下一次操作，将这个像素与矩形的边界进行比较，直至找到第一个在裁剪区内的像素，由此可知线已经通过它与一条裁剪边的交点进入了裁剪区域。然后，逐个像素地回溯以找到进入裁剪区域的第一个像素，再进行一般的扫描转换。同理，也可以类似地找到线在裁剪区内的最后一个像素，或者将检测每个像素作为扫描转换的循环操作中的一部分，并且一旦检测不成立就停止扫描转换。一般地，其间隔选择为8是比较合适的，因为在进行检测的次数和回溯的像素个数之间，这是一个比较好的折中。

对于用浮点数进行操作的图形包来说，最好是在浮点坐标系中进行解析的裁剪，然后再对裁剪后的图元进行扫描转换。此时，要注意对判定变量进行正确的初始化，如在3.2.3节中我们处理线时所做的那样。对于整型的图形包，如SRGP，要选择是预先裁剪再扫描转换还是在扫描转换中进行裁剪。因为对线和多边形进行解析的裁剪相对要容易一些，因此对它们常常是先裁剪再扫描转换。但是对其他图元，在扫描转换过程中进行裁剪就要快一些。而在一个浮点型的图形包中，最常见的是在浮点数坐标系中进行解析的裁剪，然后再调用下层的扫描转换软件来生成裁剪后的图元。这种整型的图形软件随后可以相对于矩形（甚至任意形状）窗口边界在光栅上再进行一次裁剪。因为解析地裁剪图元对整型图形包以及2D和3D的浮点型图形包都是可用的，我们就在本章讨论一些基本的解析裁剪算法。

3.9 线段裁剪

本节讨论用矩形裁剪线的解析方法；裁剪其他图元的算法，将在后面的几节中讨论。尽管有一些关于矩形裁剪和多边形裁剪的特别算法，但是值得注意的是，对SRGP的由线条组成的图元（例如，折线、非填充的矩形和多边形）的裁剪，可以通过反复运用裁剪线的方法来实现。特别是，因为圆和椭圆可以用一系列很短的线进行分段的线性逼近，因此，它们的边界可以当作是一个折线或多边形来进行裁剪和扫描转换。在一些系统中，二次曲线是由参数多项式的比率表示的（见第9章），这种表示很适于递增的分段线性逼近，这样，它们可方便地运用裁剪线的算法。用一个矩形裁剪一个矩形的结果，最多是一个矩形。用一个矩形裁剪一个凸多边形的

结果，最多也是一个凸多边形。但是，裁剪一个凹多边形就可能会得到多个凹多边形。用一个矩形裁剪一个圆或椭圆的结果，最多是4个弧段。

线与矩形裁剪区域（或者任意凸多边形的裁剪域）的交，通常是被裁剪成一条线段；线若与矩形裁剪框的边重合，也认为是在裁剪框内并且被显示出来。图3-25列出了裁剪线的几种情况。 101

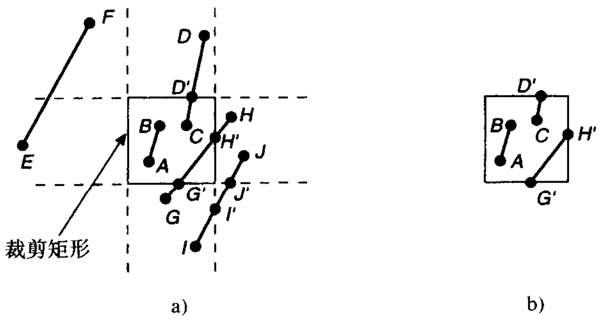


图3-25 裁剪线的几种情况

3.9.1 裁剪端点

在讨论裁剪线以前，我们先看看简单的裁剪单个点的问题。如果裁剪矩形的x坐标的边界是x_{min}和x_{max}，而y坐标的边界是y_{min}和y_{max}，那么一个位置为(x, y)的点在裁剪矩形内，就必须满足下面的4个不等关系：

$$x_{min} \leqslant x \leqslant x_{max}, y_{min} \leqslant y \leqslant y_{max}$$

如果不能满足这4个不等关系中的任意一个，这个点就在裁剪矩形外。

3.9.2 利用求解联立方程组的线段裁剪

裁剪一条线，我们只需考虑它的端点，而不必关心它无穷的内部点。如果一条直线的两个端点都在裁剪矩形内（例如，图3-25中的线AB），那么该线完全在裁剪矩形内，因此就**“简单接受”**。如果一个端点在外，而另一个在内（比如图中的线CD），则线与裁剪矩形相交，我们必须计算出交点。如果两个端点都在外，则线可能与裁剪矩形相交，也可能不相交（图中的线EF, GH, IJ），此时，我们必须做进一步的计算以决定是否有相交的部分，如果有，则需决定它们在什么位置。

一种简单的裁剪线的方法是将裁剪矩形的4条边都与线求交，看是否有交点在这些边上。如果有，这条线就与裁剪矩形相交，且有一部分在裁剪矩形内。所以，对于每一条线和每一条裁剪矩形的边，我们选取两条在数学上具有无穷长度的线，它们分别与被检测的线和裁剪矩形的边重合。然后，求这两条无穷长度的线的交点，并判断交点是否在“里面”，也就是说，它是否既在被检测的线上，又在裁剪矩形的边上。如果是，则被检测线与裁剪矩形相交。在图3-25中，交点G'和H'在里面，而I'和J'在外面。 102

我们运用这个方法时，对每一个 <边，线> 对，要用乘法和除法求解两个联立方程。尽管可以利用解析几何中关于线的方程表达式，但那是描述无穷长度的线的，而在图形学和裁剪中，我们处理的是有穷长度的线（在数学上，它们叫线段）。再说，线的斜距式不能处理垂直线这种特殊情况，比如裁剪矩形中直立的边。线段的参数式表达式可以解决这两个问题：

$$x = x_0 + t(x_1 - x_0), y = y_0 + t(y_1 - y_0)$$

此方程组刻画了从（x_0, y_0）到（x_1, y_1）的有向线段上的点（x, y），其中，参数t的变化范围是[0, 1]。只需要简单地将参数t替换掉，就能回到常见的线性方程。对关于边和线段的两组联立

方程组的参数t_{edge}和t_{line}进行求解，并检查t_{edge}和t_{line}是否都在[0, 1]之中。如果是，则其交点位于这条边和这条线段中，它便是线与裁剪矩形的交点。特别是，在解联立方程前，必须检测是否有线与裁剪矩形的边平行这种特殊情况。总之，这种直接计算的方法要做许多计算和检测，它的效率不高。

3.9.3 Cohen-Sutherland 线裁剪算法

Cohen-Sutherland算法是一种很有效的裁剪算法，它在初始化时，对线做一些检测，以决定是否可以不进行求交计算。该方法首先检测线的两个端点，看它们是否都在裁剪矩形内。若是，则该线便完全在裁剪矩形内；否则就进行下一步的区域检测。例如，在图3-25中的线*EF*，只需对它做两次简单的x坐标的比较，就知道它的两个端点的x坐标均小于x_{min}，这样，它就在裁剪矩形左边的区域（即位于左端的边所定义的外半平面内）。于是，线*EF*就能被**简单地拒绝**，而不必进行裁剪和显示了。类似地，对于x_{max}所定义的右边的区域、y_{min}定义的下边的区域和y_{max}定义的上边的区域，只要线段的两个端点都同在某一个区域，就知道这条线必不在裁剪矩形内，可以简单地裁剪掉。

若通过上面的步骤不能将线段简单地接受或拒绝，就由一条裁剪边将该线段分成两个部分，其中有一个部分可以简单地拒绝。如此循环地对线段进行裁剪和检测是否简单地接受或拒绝的操作，直至剩余的部分完全在裁剪矩形内或能被简单地拒绝。这个算法对两种常见的情况特别有效。第一种情况是裁剪矩形非常大，它几乎覆盖了整个显示区域，此时，绝大部分图元都能简单地接受。第二种情况是裁剪矩形很小，几乎可以简单地拒绝所有的图元，这在运用鼠标进行图元选择的工作时会遇到，此时，环绕鼠标的一个小矩形（称为**拾取窗口**）用来裁剪图元，
103 以判断哪些图元位于鼠标**拾取点**附近的小范围（矩形）内（见7.11.2节）。

在执行简单地接受或拒绝的检测操作时，我们将裁剪矩形的边延长并由此将平面划分成9个区域（见图3-26）。每个区域分配一个4位的代码，这个代码是根据裁剪矩形的边所定义的外半平面和各个区域的相对关系来决定的。外码中的每一位被置成1（真）或0（假）；这样的4位代码对应以下情况：

第1位，上端边所定义的外半平面，在上端边上方，$y>y_{max}$

第2位，下端边所定义的外半平面，在下端边下方，$y<y_{min}$

第3位，右端边所定义的外半平面，在右端边右方，$x>x_{max}$

第4位，左端边所定义的外半平面，在左端边左方，$x<x_{min}$

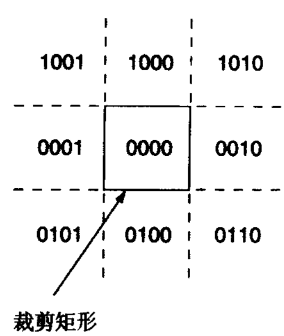


图3-26 区域的外码

例如，对既在裁剪矩形的上方又在它的左方的区域，它位于上端边和左端边所定义的两个外半平面内，它的代码就是1001。基于下述的原理，我们可以得到一个计算外码的很有效的方法，即各位的值可分别根据$(y_{max}-y)$、$(y-y_{min})$、$(x_{max}-x)$、$(x-x_{min})$的符号决定。然后，线段的每一个端点根据它所在的区域被赋予该区域的代码。根据线段的两个端点的代码，我们可以判断线段是完全在裁剪矩形里面还是在某条边所定义的外半平面内。如果这两个端点的代码中的每一位都是0，那么这条线段完全在裁剪矩形内。当然，如果两个端点都位于某条边所定义的外半平面内，比如图3-25中的线*EF*，则它的两个端点的代码在相应于这条边的位上都会被置成1，以表示这些点位于这条边所定义的外半平面内。对于线*EF*，它的外码分别是0001和1001，由第4位上的值可知该线段位于左端边所定义的外半平面内。因此，对这两个代码进行按位“与”的逻辑操作，如果结果不为0，则这条线段可以被

简单拒绝。

如果一条线段不能简单接受或拒绝，我们必须将它分成两段，再进行判断以去掉其中的一段或二段。此时，我们用与这条线相交的边来划分这条线，并将位于这条边所定义的外半平面内的那一段去掉。在检测哪条边与这条线相交时，边的选择次序可以是任意的，但在整个算法的运行过程中，这个次序必须保持不变。下面，我们将采用生成其外码的次序：由上到下，再由右到左。外码的一个重要特点是它的非零位对应于这条线会相交的边：如果一个端点位于一条边所定义的外半平面，但这条线又不能简单拒绝，则它的另一个端点必定位于这条边所定义的内半平面内，因此，这条线一定与这条边相交。所以，算法就选择一个在外面的点，并根据该点的外码中非零的位来决定一条裁剪边；所选择的边是在“由上到下，再由右到左”次序中遇到的第一条边，即外码中最左的非零位。

算法按下列方式运行。计算线段的两个端点的外码，并检测它们是否能简单接受或拒绝。如果不能，我们就选择一个在外面（至少有一个点在外面）的点，然后检测它的外码找到一条会与该线相交的边，并求出交点。随后，我们去掉从这个外面点到交点的这一段，并将交 104
点作为裁剪后的线段的一个新的端点。最后为这个新的端点计算其外码，并准备进行下一次循环操作。

例如，考察图3-27中的线段AD。点A的外码是0000，点D的外码是1001。这条线既不能简单接受也不能简单拒绝。因此，算法选择外部点D，因为它的外码显示这条线会与裁剪矩形的上端边和左端边相交。根据我们的检测次序，我们首先选择上端边将线AD裁剪成线AB，并计算出B的外码是0000。在下一次循环操作时，我们运用简单接受或简单拒绝的测试，就能简单接受AB并显示它。

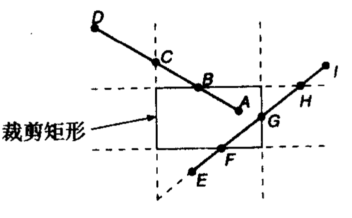


图3-27 裁剪线的Cohen-Sutherland算法示意图

裁剪线EI需要多次循环操作。第一个端点E的外码是0100，因此算法选择它作为外部点，并根据其外码知道这条线首先相交的边是下端边，由此边将EI裁剪为FI。在第二次循环操作时，不能简单接受或拒绝FI。因为端点F的外码是0000，所以算法选择外码为1010的外部点I。此时所选择的相交边是上端边，由此边裁剪出线FH。H的外码确定为0010，所以，第三次循环操作所用的裁剪边是右端边，并裁剪出线FG。在第四次循环操作时，这条线可以简单接受，因此循环结束，并显示这条线。如果开始时我们选择I作为外部点，则会得到另一种选取裁剪边的次序：根据它的外码，我们将首先裁剪上端边，然后是右端边，最后是下端边。

在程序3-7的代码中，我们用以域为成员的C结构来表示其外码，这比用一个数组表示其外码更自然一些。在此，我们用一个子程序对这些外码进行二进制的合成运算。为改善性能，我们当然要对这些代码进行排序。

程序3-7 Cohen-Sutherland线裁剪算法

```
typedef struct {
    unsigned all;
    unsigned left:4;
    unsigned right:4;
    unsigned bottom:4;
    unsigned top:4;
} outcode;
```

```
void    CohenSutherlandLineClipAndDraw(float x0, float y0, float x1, float y1,
                float xmin, float xmax, float ymin, float ymax, int value)
/* 裁剪从(x0, y0)到(x1, y1)的线的Cohen-Sutherland裁剪算法 */
/* 其裁剪矩形的对角顶点为(xmin, ymin)和(xmax, ymax) */
{
    boolean   accept, done;
    outcode outcode0, outcode1, outcodeOut;
    float   x, y;

    accept = FALSE;
    done = FALSE;
    outcode0 = CompOutCode(x0, y0, xmin, xmax, ymin, ymax);
    outcode1 = CompOutCode(x1, y1, xmin, xmax, ymin, ymax);
105 do {

        if (outcode0.all == 0 && outcode1.all == 0) {
            accept = TRUE;
            done = TRUE;
        } else if ((outcode0.all & outcode1.all) != 0)
            done = TRUE;        /* 逻辑“与”为真，所以简单拒绝，并结束算法 */
        else {
            if (outcode0.all != 0)
                outcodeOut = outcode0;
            else
                outcodeOut = outcode1;
            if (outcodeOut.top) {               /* 在裁剪矩形的上端边划分线段 */
                x = x0 + (x1 – x0) * (ymax – y0) / (y1 – y0);
                y = ymax;
            } else if (outcodeOut.bottom) {   /* 在裁剪矩形的下端边划分线段 */
                x = x0 + (x1 – x0) * (ymin – y0) / (y1 – y0);
                y = ymin;
            } else if (outcodeOut.right) {  /* 在裁剪矩形的右端边划分线段 */
                y = y0 + (y1 – y0) * (xmax – x0) / (x1 – x0);
                x = xmax;
            } else if (outcodeOut.left) {       /* 在裁剪矩形的左端边划分线段 */
                y = y0 + (y1 – y0) * (xmin – x0) / (x1 – x0);
                x = xmin;
            }
            if (outcodeOut.all == outcode0.all) {
                x0 = x;
                y0 = y;
                outcode0 = CompOutCode(x0, y0, xmin, xmax, ymin, ymax);
            } else {
                x1 = x;
                y1 = y;
                outcode1 = CompOutCode(x1, y1, xmin, xmax, ymin, ymax);
            }
        }                       /* 划分 */
    } while (!done);
    if (accept)
        MidpointLineReal(x0, y0, x1, y1, value);   /* 浮点坐标的版本 */
}

outcode CompOutCode (float x, float y,
            float xmin, float xmax, float ymin, float ymax)
{
    outcode code;
    code.top = 0, code.bottom = 0, code.right = 0, code.left = 0, code.all = 0;
    if (y > ymax) {
        code.top = 8;
        code.all += code.top;
```

```
        } else if (y < ymin) {
            code.bottom = 4;
            code.all += code.bottom;
        }

        if (x > xmax) {
            code.right = 2;
            code.all += code.right;
        } else if (x < xmin) {
            code.left = 1;
            code.all += code.left;
        }
        return code;
    }
```

106

通过避免重复计算斜率，我们可以将算法的效率稍微提高一些（见习题3.22）。不过即使有此改进，这个算法也不是效率最高的。因为检测和裁剪是按照一个固定的次序运行的，该算法有时会做一些不必要的裁剪。当线段与矩形的边的交点是“**外部交点**”时就是这种情况：也就是说，交点不在裁剪矩形的边界上（比如，图3-27中在线EI上的点H）。相比较而言，Nicholl、Lee和Nicholl[NICH87]的算法就可以避免计算外部交点，其方法是将平面划分成许多的区域，这将在[FOLE90]的第19章中讨论。在此介绍的简单的Cohen-Sutherland算法的一个优点是：它可以直接拓展应用于三维正交视见体，这将在6.6.3节中讨论。

3.9.4 参数化的线裁剪算法

Cohen-Sutherland算法可能至今仍是最常用的线裁剪算法，因为它提出得很早并且得到了广泛的传播。1978年，Cyrus和Beck提出了一个完全不同但更有效的线裁剪算法[CYRU78]。Cyrus-Beck算法可以用来在平面上由一个矩形或一个任意形状的凸多边形对一条二维的线进行裁剪，或者在空间由一个任意的凸多面体对一条三维的线进行裁剪。后来梁友栋和Barsky独立地提出了一个更有效的参数化线裁剪算法，特别是，该算法在处理二维或三维的直立矩形裁剪区域时非常快[LIAN84]。除了发掘简单的裁剪边界的优点外，他们还为一般的裁剪区域引入了更有效的简单拒绝的检测。在此，我们从初始的Cyrus-Beck算法开始来介绍参数化的裁剪操作。但是，因为我们关注的只是直立的矩形裁剪矩形，在讨论的最后，我们将Cyrus-Beck算法归约为更有效的梁友栋–Barsky算法。

在Cohen-Sutherland算法中，对于不能简单接受或拒绝的线都要计算这条线与一条裁剪边的交点(x, y)，在求解过程中，是将垂直或水平的裁剪边的x坐标或y坐标值代入线的方程进行计算。然而，在线的参数方程中，求取的是参数t，它对应于裁剪边所在的无穷长的线与被裁剪线的交点。广义地说，4条裁剪边都会与被裁剪线相交，这样，就需要算出4个t值。然而只需 107
经过一系列的简单比较，以判断这4个t值中的哪些（如果存在）对应于真正的交点。随后，只对一个或两个真正的交点坐标(x, y)进行计算。显然，与Cohen-Sutherland中的求交算法相比，该算法会节约许多时间，因为它不必求线段与各个裁剪边的交点，可减少循环操作的次数。再说，在一维参数空间中的计算要比三维坐标空间中的计算简单。在Cyrus-Beck算法的基础上，梁友栋和Barsky做了进一步的改进，一旦一个t值计算出来就进行检测，并裁剪掉一部分线段，而不必等到4个t值都计算出来后再进行检测。

Cyrus-Beck算法的基础是下面的两条线求交的公式。图3-28中显示了一条裁剪边E_i和这条边

向外的法向N_i（也就是指向裁剪矩形外的方向）⊖，以及从P_0到P_1这条将被该边裁剪的线段。在求交点时，可能要对这条裁剪边或线段进行延长。

如同前面所介绍的，这条线的参数化表达式是：

$$P(t) = P_0 + t(P_1 - P_0)$$

其中，在P_0处$t = 0$，而在P_1处$t = 1$。现在，我们在边E_i上任取一点P_{E_i}，并考察从P_{E_i}到线段P_0P_1上3个点的3个向量，这3个点是：将被确定的交点、位于裁剪边所定义的内半平面内的端点和位于裁剪边所定义的外半平面内的端点。通过计算点积$N_i \cdot [P(t) - P_{E_i}]$的值，我们可以知道各个点位于哪个区域。对于在内半平面内的点，其点积是负数；对于在边上的点，点积是零；而对于在外半平面内的点，其点积是正数。关于一条边的内外半平面的定义对应于沿裁剪区域边的逆时针顺序。这是我们在本书中将始终遵循的一条常规。现在，我们

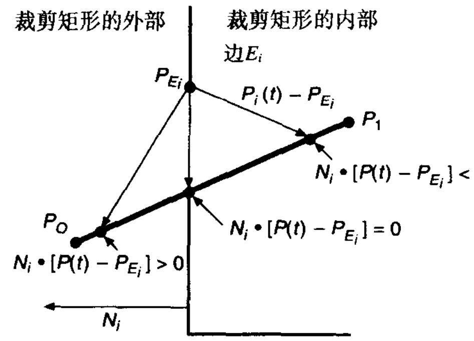


图3-28 外部点、内部点和裁剪框边界上的点的点积

108 可以用下面的方程来求解线P_0P_1与边相交时的参数t的值：

$$N_i \cdot [P(t) - P_{E_i}] = 0$$

首先，替换掉$P(t)$后得到

$$N_i \cdot [P_0 + t(P_1 - P_0) - P_{E_i}] = 0$$

然后，合并同类项，并运用点积的分配定律，我们得到

$$N_i \cdot [P_0 - P_{E_i}] + N_i \cdot t[P_1 - P_0] = 0$$

设从P_0到P_1的向量为$D = (P_1 - P_0)$，于是t的解为：

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D} \tag{3-1}$$

注意，只有在表达式中除数不为零时，我们才能得到一个有效的t值。

为保证这一点正确，算法要做如下检测：

$N_i \neq 0$（即法线不能为0；只有出现错误时，才会发生这种情况），

$D \neq 0$（即$P_1 \neq P_0$），

$N_i \cdot D \neq 0$（也就是，边E_i和从P_0到P_1的线段不平行。如果它们平行，就不会与这条边有单个交点，如此，算法就得对这种情况进行进一步的讨论）。

式(3-1)可以用来计算线段P_0P_1与裁剪矩形的每一条边的交点。在做这种计算时，我们要为每条边确定法向和边上的任意一个点P_{E_i}（比如说，这条边的一个端点），随后，这些值可用来求该边与所有线段的交点。假设为一条线段已求出了4个参数t值，下一步就是要判断其中哪些值确实相应于裁剪框的边与线段的真正的交点。首先，在区间[0, 1]之外的t值可以去掉，因为，它位于线段P_0P_1之外。然后，我们需要判断交点是否在裁剪边界上。

从图3-29中线1的情况得到启发，我们可以简单地对余下的t值进行排序，并选择中间的t值来求取交点。但我们怎样将线1的情况与线2的情况区别开来呢？在线2的情况中，这条线与裁剪矩形没有任何相交的部分，而中间的t值相应的点也不在裁剪边上。再者，线3上的4个交点，

⊖ Cyrus和Beck用的是向内的法线，但我们倾向于用向外的法线，这是为了与三维空间中定义平面的法线保持一致，因为它们是向外的。因此，我们这里所用的方式与Cyrus和Beck的差别只是符号的检测不同。

哪些又在边界上呢？

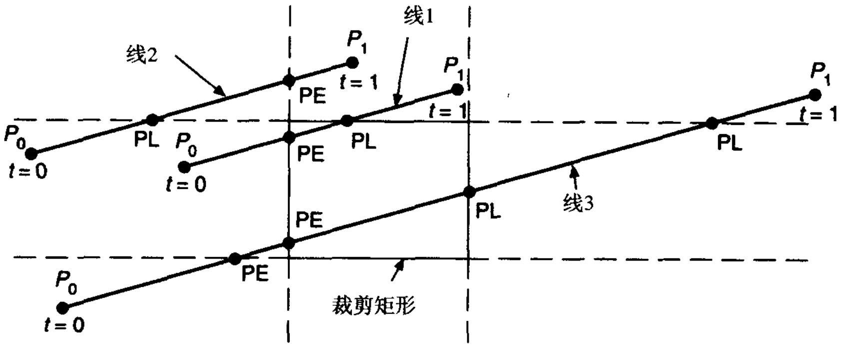


图3-29 沿着裁剪矩形的对角方向分布的一些线段

根据下面的原则，图3-29中所示的交点可以相对于裁剪矩形，形象地分为“可能进入点”（PE）和“可能离开点”（PL）。这些原则是：从P_0往P_1移动时，如果跨过一条边就进入该边所定义的内半平面，这个交点就是一个PE；否则，如果是离开该边所定义的内半平面，则该交点是一个PL。我们注意到，根据这样的区分，一条线与裁剪矩形的两个靠里面的交点会有不同的标志。

正式地说，根据线P_0P_1与N_i的夹角可以将交点分成PE和PL两类：如果夹角小于90°，交点就是PL的；夹角大于90°，交点就是PE的。这些信息都隐含在N_i和P_0P_1的点积的符号里了：

$$N_i \cdot D<0 \Rightarrow \text{PE（夹角大于90°）},$$

$$N_i \cdot D>0 \Rightarrow \text{PL（夹角小于90°）}。$$

注意$N_i \cdot D$只是式(3-1)中的除数，这意味着，在计算t的过程中，我们就能很容易地知道交点是什么类型的。根据这样的分类，由图3-29中的线3可以得到算法的最后步骤。也就是，我们必须选择一个能确定最后被裁剪的线的（PE, PL）对。对于穿过P_0P_1的无穷长的线，其在裁剪矩形内的线段是由一个PE点和一个PL点确定的，其中，PE点有最大t值t_E，而PL点有最小t值t_L。于是，裁剪出的线段就由（t_E，t_L）范围确定了。但是，因为我们关心的是线段P_0P_1上被裁剪出来的部分，而不是无穷长的线，因此，我们要对这个范围做进一步的修改使得于t_E的下界必是$t = 0$，而t_L的上界是$t = 1$。如果$t_E > t_L$会怎样呢？这正是线2的情况。这时意味着线段P_0P_1与裁剪矩形没有相交部分，因此，整条线段都被拒绝。真实交点的t_E和t_L的值将用来计算相应的x和y的坐标值。

程序3-8给出了关于直立矩形的完整裁剪算法的伪代码。[FOLE90]中的图3-45给出了[LIAN84]算法改编后的完整的代码。

程序3-8 Cyrus-Beck参数化线裁剪算法的伪代码

```
{
  预计算N_i，并为每条边选择一个P_Ei点；

  for（每条要被裁剪的线段）
     if (P_1 = P_0)
        退化成一个点的线被作为一个点进行裁剪；
     else {
        t_E = 0;
        t_L = 1;
        for（与一条裁剪边相关的每个可能的交点）
           if (N_i · D != 0) {          /* 不考虑与裁剪边平行的边 */
              计算t;
```

109~110

```
                    用Ni · D的符号来区分PE和PL;
                    if (PE) tE = max(tE, t);
                    if (PL) tL = min(tL, t);
                }
            }
            if (tE > tL)
                return nil;
            else
                P(tE)和P(tL)作为真正的裁剪交点;
        }
    }
}
```

概括地说，如果对于外码检测的开销不大（比如，在汇编语言中运行位操作），并且对大部分线段可以简单拒绝或接受，Cohen-Sutherland算法是很有效的。当要对很多线进行裁剪时，参数化的线裁剪方法更好一些，因为它尽量避免了求交点坐标的计算，并且是基于参数值进行检测的。但是，在Cohen-Sutherland算法中本可以简单接受的端点，在参数化算法中却要进行参数计算。梁友栋–Barsky算法比Cyrus-Beck算法更有效的原因是，对于不会与裁剪矩形相交的线段，它所附加的简单拒绝的检测可以避免计算所有4个参数值的工作。对于不在非可见半平面内的线段，Cohen-Sutherland算法不能简单地拒绝，而必须重复多次裁剪才能判断出来，而梁友栋-Barsky算法中关于拒绝的检测就可以做到这一点。一般地说，Nicholl等人提出的算法要比Cohen-Sutherland算法和梁友栋-Barsky算法都好，但它不能像参数化裁剪那样推广到三维。对Cohen-Sutherland算法的加速处理在[DUVA90]中讨论。

3.10 圆的裁剪

用一个矩形裁剪一个圆，我们首先要用下一节将介绍的多边形裁剪算法做一个简单的拒绝或接受的检测，即求圆所在的方形区域（即边长为圆的直径的正方形）与裁剪矩形的交。如果圆与裁剪矩形相交，我们就将圆均分成4个部分并对每个部分进行简单拒绝/接受的检测。这些检测可能会使得圆要被继续分成一些八分之一部分并对这些部分进行检测。然后，通过并行地求解圆和边的方程组，我们可以解析地求得圆和边的交点。最后，扫描转换所计算的圆弧。在此，对算法要进行适当的初始化，所用的起始点和结束点就是所求的交点（进行了适当的取整处理）。如果扫描转换很快，或者圆不是太大，将圆边界上的像素一个一个地相对于裁剪矩形的边界进行检测并截剪的操作，在写像素之前完成，可能更加有效。在任何情况下，范围检测都是很有用的。如果是对圆进行填充，可以对每个跨段进行裁剪再填充其中的像素，这样，就
111 不必将跨段中的像素相对于裁剪边界进行逐个的检测了。

3.11 多边形裁剪

如图3-30所示，裁剪一个多边形的算法必须处理很多不同的情况。特别值得注意的是图3-30a中的情况，一个凹多边形被裁剪成了两个独立的多边形。总之，裁剪工作是相当复杂的。多边形的每一条边必须相对于裁剪矩形的每一条边进行检测。裁剪过程中，可能要增加新的边，而对于已有的边，要判断是舍弃、保留还是剖分。裁剪一个多边形可能会得到多个多边形。我们需要一种结构很好的方法来处理所有这些情况。

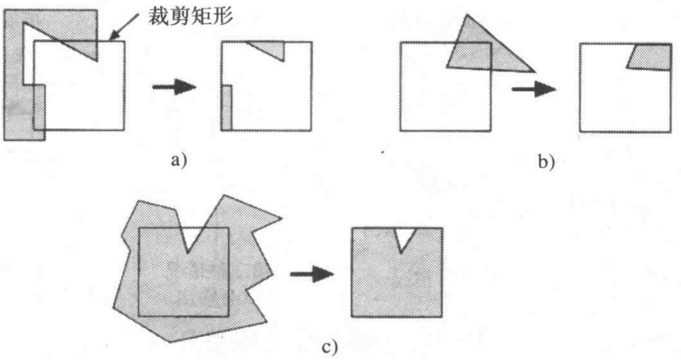


图3-30 多边形裁剪的例子。a) 产生了多个单元，b) 简单的凸多边形情况，c) 有多条外凹边的情况

Sutherland-Hodgman多边形裁剪算法

Sutherland-Hodgman多边形裁剪算法[SUTH74b]采用了分而治之的策略：它先解决一些简单而明确的问题，然后，综合起来就可以解决全部的问题。这个简单问题就是用一条无穷长的裁剪边来裁剪一个多边形。对于裁剪矩形，就是用其4条边对多边形进行连续的裁剪操作（见图3-31）。

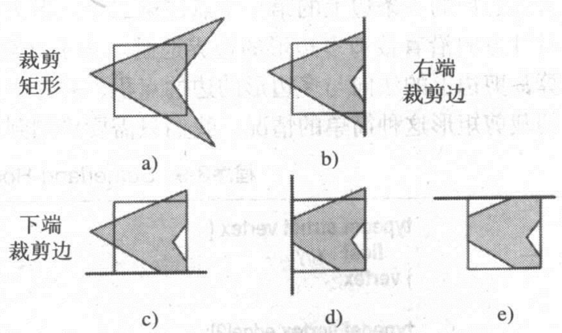
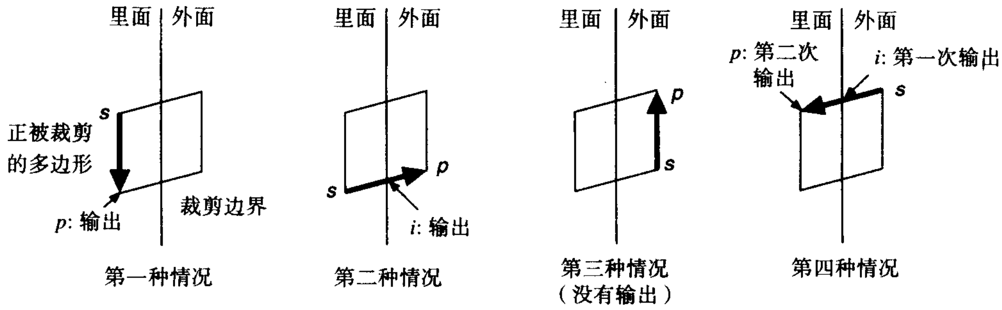


图3-31 多边形裁剪沿着裁剪边依次进行。a) 裁剪前；b) 在右边裁剪；c) 在底边裁剪；d) 在左边裁剪；e) 在顶边裁剪，多边形被裁剪完毕

值得注意的是：裁剪多边形的策略与Cohen-Sutherland裁剪线的算法是不同的。裁剪多边形时，要连续地对4条边进行裁剪，而裁剪线时，是检测外码以判断线段跨越了哪条边，并且只在需要时才进行裁剪。实际上，Sutherland-Hodgman算法的应用范围很广：任何一个凸的或凹的多边形都可以相对于一个凸的裁剪多边形进行裁剪；在三维空间中，多边形可以相对于由平面片构成的多面体进行裁剪。该算法的输入参数是多边形的一串顶点$v_1, v_2, \cdots, v_n$。在二维空间，多边形的边是根据其顶点依次连接生成的，即从v_i到v_{i+1}是一条边，而最后一条边是从v_n到v_1。将多边形相对于一条无穷长的裁剪边进行操作，其输出结果是关于裁剪后的多边形的一串顶点。随后，对刚得到的多边形，相对于第二条裁剪边进行裁剪。如此继续，直至相对于所有裁剪边进行了裁剪操作。

该算法的操作过程是沿着多边形的边从顶点v_n移动到v_1，再顺序移动回到v_n，在每一次移动时，都检测连续的两个顶点与裁剪边的相互关系。在每一步，对于裁剪后的多边形的顶点序列，可能会增加一个顶点或两个顶点，也可能不会增加顶点。此时，要分析4种情况，如图3-32所示。

在图3-32中，考察多边形的从顶点s到p的边。假设在上一次循环操作中已经处理了起始点s。在第一种情况时，多边形的边完全在裁剪边的里面，所以，顶点p被加到输出的顶点序列中。在第二种情况时，因为多边形的边与裁剪边界相交，所以将交点i输出。在第三种情况时，因为两个顶点都在边界的外面，所以没有顶点输出。在第四种情况时，交点i和顶点p都加到输出的顶点序列中。



112
~
113

图3-32 多边形裁剪的4种情况

程序3-9中的函数SutherlandHodgmanPolygonClip()的输入是顶点的一个数组*inVertexArray*，而其输出的是所产生的另一个顶点数组*outVertexArray*。为了简化程序，我们没有对数组进行越界检测，同时我们用函数Output()将一个顶点放入*outVertexArray*数组中。函数Intersect()计算由顶点*s*和*p*确定的多边形的边与一条裁剪边的交点，裁剪边是由裁剪多边形边界上两个顶点定义的。如果顶点在裁剪边界的里面，函数Inside()就返回TRUE。在此，“里面”的意思是“当从裁剪多边形的一条边上的第一个点往第二个点看去时，顶点位于该裁剪边的左方”。这相当于以逆时针方向沿着裁剪多边形的边界巡游。为了计算一个点是否在一条裁剪边的外面，我们可以计算裁剪边上的法向与多边形的边的点积，并考察它的符号，如同3.9.4节中所介绍的。（对于直立的裁剪矩形这种简单的情况，我们只需要检测到裁剪边界的水平距离或垂直距离的符号。）

程序3-9 Sutherland-Hodgman多边形裁剪算法

```
typedef struct vertex {
    float  x, y;
} vertex;

typedef vertex edge[2];
typedef vertex vertexArray[MAX];          /* MAX是一个声明的常量 */

void  Intersect(vertex first, vertex second, vertex *clipBoundary,
              vertex *intersectPt)
{
    if (clipBoundary[0].y == clipBoundary[1].y) {        /* 水平 */
        intersectPt->y = clipBoundary[0].y;
        intersectPt->x = first.x + (clipBoundary[0].y – first.y) *
            (second.x – first.x) / (second.y – first.y);
    } else {                                             /* 垂直 */
        intersectPt->x = clipBoundary[0].x;
        intersectPt->y = first.y + (clipBoundary[0].x – first.x) *
            (second.y – first.y) / (second.x – first.x);
    }

    }


    boolean  Inside(vertex testVertex, vertex *clipBoundary)
    {
        if (clipBoundary[1].x > clipBoundary[0].x)          /* 下 */
            if (testVertex.y >= clipBoundary[0].y) return TRUE;
        if (clipBoundary[1].x < clipBoundary[0].x)          /* 上 */
            if (testVertex.y <= clipBoundary[0].y) return TRUE;
        if (clipBoundary[1].y > clipBoundary[0].y)          /* 右 */
```

```
        if (testVertex.x <= clipBoundary[1].x) return TRUE;
    if (clipBoundary[1].y < clipBoundary[0].y)            /* 左 */
        if (testVertex.x >= clipBoundary[1].x) return TRUE;
    return FALSE;
}

void  Output(vertex newVertex, int *outLength, vertex *outVertexArray)
{
    (*outLength)++;
    outVertexArray[*outLength – 1].x = newVertex.x;
    outVertexArray[*outLength – 1].y = newVertex.y;
}

void  SutherlandHodgmanPolygonClip(vertex *inVertexArray,
        vertex *outVertexArray, int inLength, int *outLength, vertex *clip_boundary)
{
    vertex s, p, i;
    int    j;

    *outLength = 0;
    s = inVertexArray[inLength – 1];     /* 从inVertexArray中最后的一个顶点开始 */
    for (j = 0; j < inLength; j++) {
        p = inVertexArray[j];   /* 此时的 s 和 p 相应于图3-33中的顶点 */
        if (Inside(p, clip_boundary)) {                  /* 第1种和第4种情况 */
            if (Inside(s, clip_boundary))
                Output(p, outLength, outVertexArray);    /* 第1种情况 */
            else {                                       /* 第4种情况 */
                Intersect(s, p, clip_boundary, &i);
                Output(i, outLength, outVertexArray);
                Output(p, outLength, outVertexArray);
            }
        } else if (Inside(s, clip_boundary)) {           /* 第2种和第3种情况 */
            Intersect(s, p, clip_boundary, &i);          /* 第2种情况 */
            Output(i, outLength, outVertexArray);
        }                                          /* 在第3种情况下，没有操作 */
        s = p;                                 /* 转到下一对顶点继续运行 */
    }
}
```

Sutherland和Hodgman指出了对算法如何进行组织就能保证算法能够反复操作[SUTH74b]。一旦输出一个点，算法就用此点来调用自身。再相对于下一条裁剪边界进行裁剪操作。这样，对于已经裁剪了一部分的多边形，就不需要临时的存储空间了：实际上，这个多边形是在裁剪算法的操作“流水线”中穿行。每一步都可以作为一种不需要附加缓冲空间的特殊硬件来实现。这种特点（以及它的普适性）使得这个算法能够适用于当前的硬件实现。但是，在算法执行中可能会在裁剪矩形的边界上引入新的边。考察图3-30a中的情况，连接三角形的左上点和矩形的左上点就引入了一条新的边。在后处理时，要删除这些边。 114~115

3.12 生成字符

3.12.1 定义和裁剪字符

定义字符有两种基本的方法。最一般而又计算开销很大的方法是将每个字符定义为一条曲线或多边形的轮廓，然后在需要时进行扫描转换。在此，我们先讨论另一种较简单的方法：对

于给定某种字体的每一个字符，生成一个小型的矩形位图。然后，在产生字符时，只需简单地用copyPixel将字符的图像从一个称为**字体高速缓存**的屏幕外画布中复制到目标位置的帧缓存中。

如下所述，字体高速缓存实际上可以在帧缓存中。对于大多数用一个专用的帧缓存来刷新屏幕的图形系统来说，其帧缓存的空间是大于存储一幅显示图像所要的空间的。例如，一个矩形屏幕上的像素可以存储在一个方形的存储空间中，但该空间中有一条矩形状的区域是在屏幕上"不可见的"。或者，当存储空间比较大，足以容纳两个屏幕时，存储空间就分成两个部分，一部分用来对当前屏幕进行刷新，而另一部分用来写入图像，即对图像进行双缓冲的操作。因为显示控制器的copyPixel在本地图像空间中运行最快，因此，为当前显示字体服务的字体高速缓存往往是存放在这样的不可见的屏幕存储区中。有关这种不可见存储区的一种应用是，在弹出如窗口、菜单或者表格的图像时，对于临时被遮挡的屏幕内容进行保留。

字体高速缓存中存的位图往往是对印刷字体进行各种程度的放大，然后再进行扫描生成的；那么，字型设计师在需要的时候可以用一个绘图程序对每个字符位图中的各个像素进行修改。或者，字型设计师可以从一开始就用一个绘图程序来为屏幕或低分辨率的打印机专门设计字体。由于小位图在缩放时质量不是太好，因此，对于给定字体的每一个字符要定义多种尺寸大小的位图，以便提供各种大小的标准字体。而且，每个字型有它自己的一套位图。因此，在实际应用时，要为每种字体分配一个单独的字体高速缓存。

在SRGP中，对位图型的字符自动进行裁剪，这是作为copyPixel命令的一部分功能实现的。每个字符是按照逐个像素的方式裁剪到目标矩形中去的，这样我们就可以在字符位图的任何列或行上进行裁剪。若一个系统中的copyPixel命令运行比较慢，一种比较简单但快速的方法是对字符甚至一个字符串进行一种"要么全部接受，要么全部舍弃"的裁剪。只有当字符或字符串的区域可以完全接受的时候，才用copyPixel命令去写字符或字符串。即便系统中的copyPixel命令运行比较快，预先对字符串的范围进行简单的接受/拒绝检测依然是很有用的，这有利于在copyPixel操作中删除一些字符。

SRGP中是运用简单的位图字体高速缓存技术将字符一个靠一个地存储在画布上。当然这个画布很宽，但它的高度只是最高字符的高度。图3-33中显示了这个高速缓存的一部分以及几个离散的低分辨率字符的例子。每一种被调用的字体都由一个结构
116 （在程序3-10中声明的）来描述，该结构包含一个指向存储字符图像的画布的指针、字符的高度信息以及在字符串中相邻字符之间的间距。（在有些软件包中，将字符间距作为字符宽度的一部分保存，允许字符间有不同间距。）

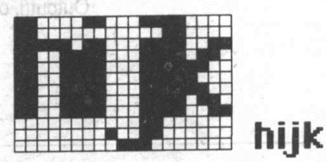


图3-33 一个字体高速缓存例子的一部分

程序3-10 字体高速缓存的类型声明

```
typedef struct charLocation {
    int leftX, width;                    /* 在字体高速缓存中图像的水平位置和宽度 */
} charLocation;

typedef struct fontCacheDescriptor {
    canvasIndexInteger cache;
    int descenderHeight, totalHeight;    /* 高度是常量，宽度是变化的 */
    int interCharacterSpacing;           /* 按照像素的个数进行度量 */
    charLocation locationTable[128];
} fontCacheDescriptor;
```

如2.1.5节中所介绍的，对于一个给定的字体，其下降部分高度和整个高度是不变的——前者是只为字符的下降部分所用的字体高速缓存中下端的像素行数，而后者则是字体高速缓存画布的高度。但另一方面，字符的宽度并不作为一个常量；这样，字符就能根据自身的大小来占有空间，而不必硬性塞进固定宽度的字符框里。画一个字符串时，SRGP是设置一个固定的字符间距，这个间距是在每种字体的描述器中给出的。用SRGP来显示文本中的各个字符，字处理程序可以显示多行文本，并且可以通过变化字符间距来适当调整各行的情况，以及在标点结束后继续填完各行，以保证各行中最右的字符在右端对齐。这包括用探知文本大小的工具来决定每个单词右边的位置，以便计算下一个单词从何处开始。显然，对于复杂的字处理应用，SRGP的字处理工具是很粗糙的，它不能用于排版程序，因为排版程序要求对单个字母的范围进行更精细的规划，以便处理一些不能进行水平对齐的情况，如上标、下标以及对文本中的一些字母进行缩放和变化的情况。

3.12.2 一种文本输出图元的实现

在程序3-11的程序中，我们揭示了SRGP文本在内部是如何实现的：给定的字符串中的每个字符都是一个一个单独放置的，而字符间距是由字体描述器中适当的项表示的。注意：对于复杂的字符处理，如在字符串中混有多种字体的情况，必须由应用程序来处理。

程序3-11 SRGP文本图元的字符定位的实现

117

```
void SRGP_characterText(point origin, char *stringToPrint,
      fontCacheDescriptor fontInfo)
/* 在当前的画布中，在何处放置字母 */
{
      rectangle fontCacheRectangle;
      char charToPrint;
      int i;
      charLocation *fp;

   /* 由应用程序定义的原点为基准，同时不包括下降部分 */
      origin.y -= fontInfo.descenderHeight;

      for (i = 0; i < strlen(stringToPrint); i++) {
         charToPrint = stringToPrint[i];
         fp = &fontInfo.locationTable[charToPrint];
   /* 在高速缓存中寻找字符所在的矩形区域 */
         fontCacheRectangle.bottomLeft = SRGP_defPoint(fp->leftX, 0);
         fontCacheRectangle.topRight = SRGP_defPoint(fp->leftX + fp->width - 1,
            fontInfo.totalHeight - 1);
         SRGP_copyPixel(fontInfo.cache, fontCacheRectangle, origin);
   /* 修改原点，使它越过刚生成的字符和一个字符间距 */
         origin.x += fp->width + interCharacterSpacing;
      }
   }
```

我们曾提到，对于显示设备或输出设备的各种不同的分辨率，利用位图技术时其字体、尺寸和字型的每一种组合都要求有一个独立的字体高速缓存。若一种字体有8种不同大小的点阵和4种字型（常规的、加粗的、斜的和加粗兼斜的）就需要32个字体高速缓存！解决存储问题的一种方法是用一种抽象的与设备无关的方式来表示字符，即保存用浮点参数定义的刻画字符轮廓的多边形或曲线，然后对这些轮廓线进行适当的变换，就能得到所需要的字符。一种称为**样条**（见第9章）的多项式函数可以提供一阶和高阶连续的光滑曲线，因此，它常用来生成文

字的轮廓线。尽管定义一个字符所要的空间多于在字体高速缓存中表达该字符所需的空间，但对一个这样所保存的字符进行适当的缩放就可以得到各种大小的字符。同理，对轮廓进行适当的错切变换就可以很快地拟合斜体字型。用完全与设备无关的方式存储字符的另一个重要优点是：对轮廓可以进行任意的平移、旋转、缩放和裁剪（或者将其本身当成一个裁剪区域）。

用样条函数刻画的字符在节省空间方面并不像所期望的那样好。例如，对一个字符而言，并不是其所有尺寸的点都能通过缩放一个抽象的形状来得到，因为字体形状的好看与否与点的大小是有关的，因此，一种字型只对有限大小的一些点是最有效的。再者，对样条曲线的文字进行扫描转换，比简单地用copyPixel命令来实现需要更多的操作，因为，与设备无关的形式必须根据当前的大小、字型和变换属性等来变换到像素坐标系。所以，字体高速缓存技术在微机上应用得很普遍，甚至在一些工作站上也使用。为同时利用这两种方法各自的优点，一种策略
118 是以轮廓的形式存储字体，但在具体应用时，就将字体变换到它们对应的位图。例如，实时地生成字体高速缓存。在[FOLE 90]的19.4节中详细讨论了如何处理样条型文本。

3.13 SRGP_copyPixel

如果只有WritePixel（写像素）和ReadPixel（读像素）这样的底层程序，SRGP_copyPixel函数可以用一个双层嵌套的**for**循环来实现对每个像素的操作。为简便起见，首先假设我们是在二值显示器上工作，并且对于不用字对齐的填写位，不必考虑底层的处理。在我们简单的SRGP_copyPixel命令的内循环中，我们对源像素和目标像素执行一次ReadPixel操作，然后根据SRGP的写模式对它们进行逻辑组合，最后用WritePixel写出这个结果。对于**replace**这种很常用的写模式，它可以作为一种特例用一个简单的内循环来实现，即只是简单地用ReadPixel读取源像素并用WritePixel写到目标像素，而不必进行逻辑操作。在确定位置的时候，运用裁剪矩形来限制要写的目标像素的区域。

3.14 反走样

3.14.1 增加分辨率

至今所画的图元有一个共同的问题：它们有锯齿形的边。这种称为**锯齿图**或**梯形图**的不好效果，是因为在扫描转换时采取了绝对的"是/否"原则，即对每个像素要么填以图元的颜色，要么就完全不改变。锯齿是**走样**现象的一种情况。减轻或除去走样情况的应用技术，称为**反走样**；运用反走样技术生成的图元或图像被称为是已反走样了的。在[FOLE90]的第14章中，从信号处理的角度来讨论了一些基本理论，以解释为什么叫走样，走样为什么会发生，以及在生成图像时如何减少或去掉走样。在此，我们只是很直观地解释SRGP的图元为什么会出现走样，并且讨论如何修改本章所介绍的线扫描转换算法以生成已反走样的线。

现在，考察运用中点算法在白色背景上画一条斜率在0和1之间单个像素宽的线。在线穿过的每一列中，该算法都选择最靠近线的一个像素并赋以该线的颜色。每当线所经过的两个相邻列上的像素不在同一行上时，在画布上所画的线就会出现一次剧烈的跳变，在图3-34a中已清晰地显示出了这一点。对于其他只能给像素赋予两种亮度值的图元，在扫

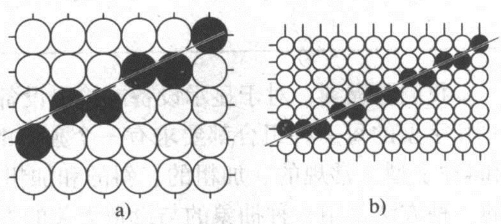


图3-34 a) 二值显示器上的标准的中点线算法，b) 同样的线画在双倍线分辨率的显示器上

描转换时也会出现同样的情况。

假设我们现在用的显示设备在水平和垂直方向上具有双倍的分辨率。如图3-34b所示，这条线穿过了两倍的列数，因此其跳变的次数也翻了一番，但每次跳变的大小，无论在x方向还是y方向上，都减少了一半。尽管这样处理后的图像要好看一些，但这种改进的代价是：空间开销、存储器的带宽以及扫描转换时间都增长到了4倍。提高分辨率的方法是一种开销很大的方法，并且，它只是淡化了锯齿问题，对此问题并没有彻底解决。在下面的几节，我们将探讨一些开销不大的反走样技术，它们依然能生成很好的图像。

3.14.2 未加权的区域采样

改善图像质量的第一种方法是基于以下的认识提出的：尽管理想的图元（如线）是没有宽度的，但我们所画的图元是有宽度的。一个扫描转换的图元在屏幕上占有有限的区域——甚至在一个显示面上最细窄的水平线或垂直线也有一个像素宽，而其他斜率的线的宽度是随着图元的不同而变化的。因此，我们可以将任何线当成一个有一定宽度的矩形，它盖住了一部分栅格，如图3-35所示。然后，根据以下准则进行操作：一条线不应该只在它所经过的每一列上选一个像素来变黑，它应该对它在每一列上所经过的每一个像素赋以一定量的亮度值。（当然，亮度值的这种变化只能在多位像素的显示器上才能反映出来。）这样，对于单个像素宽的线，只有水平线和垂直线才会在它们所属的行或列中每次只涉及一个像素；而对于其他斜率的线，在每一行或每一列中，会有多个像素被赋以各自适当的亮度值。

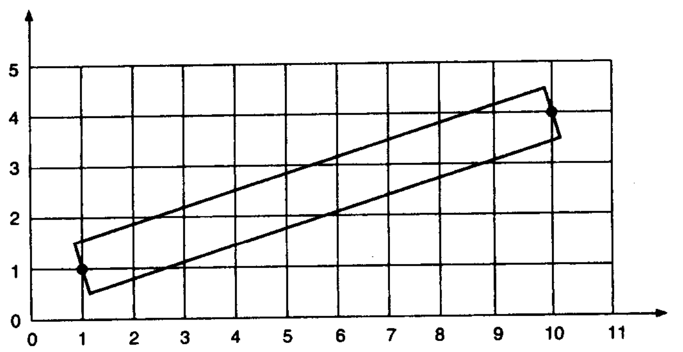


图3-35 宽度不为零的从点(1, 1)到点(10, 4)的线

但是，一个像素的几何特征是怎样的呢？它有多大？对于一条线穿过的一个像素，其亮度值该是多少？一种方便计算的假设是像素是相互间没有重叠的方形的片，可以一片挨着一片地覆盖屏幕，每一片的中心位于栅格点上，而不是像本章前面的不相交圆。（当我们说一个图元盖住了一个像素或像素的一部分时，我们是指它盖住了该像素的整片或一部分；之所以强调这个，是因为我们有时将这样的正方形当作*由像素表现的区域*。）我们还假设一条线赋给一个像素的亮度值决定于该像素的片被线覆盖的区域的大小。在黑白显示的情况下，一个被完全覆盖的像素将被赋以黑色，而被部分覆盖的像素所着的颜色为灰色，其亮度值决定于该像素被线所覆盖的范围。这种技术运用于图3-35中所示的线的情况，如图3-36所示。

对于一条在白色背景上所画的黑线，像素(2, 1)上大约70%是黑的，而像素(2, 2)上只有大约25%是黑的。与线没有相交的像素如(2, 3)就完全是白的。根据一个像素被图元覆盖的区域按比例赋予该像素相应亮度值，将淡化图元边界上刺眼的锯齿特征，得以在完全填充和完全不填充之间很平滑地过渡。这种模糊处理使得一条线从远处看质量很好，尽管它是将填充/不填充的过渡过程扩展到一行或一列上的多个像素。对覆盖区域的一种粗略计算的方法，是将像素划

分成更细的矩形的子像素栅格，然后计算表达线的矩形所覆盖的子像素的个数。例如，在线的上端边界和下端边界之间的部分（见习题3.25）。

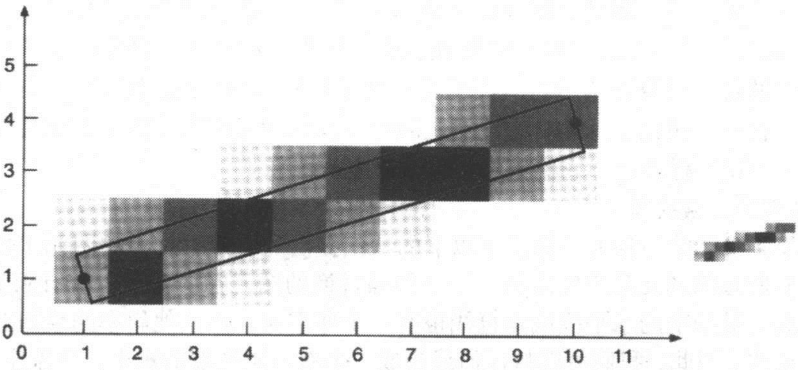


图3-36 像素的亮度与线所覆盖的面积成比例

对这种根据覆盖面积的多少来决定亮度值的方法，我们称之为**未加权的区域采样**。相比于将像素设置成完全填充或者为零的方法，这种方法明显地提高了图像质量。当然，还有一种称为**加权区域采样**的更有效的方法。为解释这两种不同形式的区域采样方法的差异，我们注意到不加权的区域采样有以下3个特点。第一个特点是，与一条线相交的像素的亮度值是随着它的中心与边的距离的增大而降低的：离图元越远，则图元对该像素亮度的影响就越小。这种关系是显然存在的，因为覆盖面积降低，亮度也降低，而当线远离像素的中心而靠近像素的边界时，覆盖面积是递减的。当线完全覆盖了像素时，亮度值将达到一个最大值；而当图元的边只在像素的边界上相切时，覆盖面积为零，因此，亮度值为零。

第二个特点是，如果图元与一个像素不相交，则图元对该像素的亮度根本就没有影响，这也就是说，图元与反映像素的正方形片不相交。第三个特点是，不管像素的中心到覆盖区域的距离如何，相同的面积大小就有相同的亮度值，也就是，只有面积的大小起作用。这样，在像素的角上的一小块面积同靠近像素中心的同样大小的面积所起的作用是一样的。

3.14.3 加权区域采样

在加权区域采样中，我们保留了未加权区域采样的前二个特点（覆盖面积降低，亮度就降低；只有当图元覆盖了像素所表达的区域，图元才能发挥作用），但我们改变了第3个特点。我们让同样面积大小的区域发挥不同大小的作用：靠近像素中心的小块区域的作用大于远离中心的同样大小区域的作用。这种变化的理论基础在[FOLE90]的第14章给出，那里，在介绍滤波原理时讨论加权区域采样。

为保留第二个特点，我们必须对像素的几何特征做以下改变。在未加权的区域采样中，对于反映一个像素的正方形片，如果图元的一条边非常靠近它的边界但与边界又不相交，则该图元对该像素的亮度没有什么影响。在新方法中，像素表示一个比正方形片稍大的圆形区域，因此，图元将与这个较大区域相交，并影响像素的亮度值。注意，这意味着与邻接像素相关的区域实际上是重叠的。

119~122 为了解释名称中“未加权”和“加权”这两个形容词的缘由，我们定义一个**权值函数**，它决定图元中给定的一小块区域dA对一个像素的亮度的影响值，它是关于dA到像素中心的距离的一个函数。对于未加权的区域采样，该函数的值是常量；而对于加权的区域采样，距离越远，则函数值越小。将权值函数当成平面上的一个函数$W(x, y)$，则对在(x, y)处的面积dA，其权值

就是在(x, y)处的高度。对于未加权的区域采样，当像素表示为正方形贴片时，W的图形就是一个方盒，如图3-37所示。

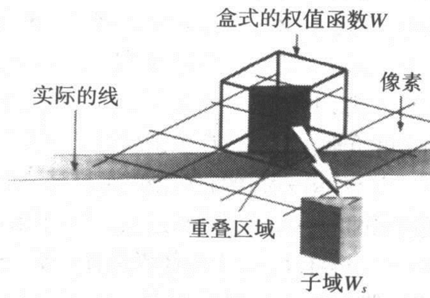


图3-37 针对正方形像素的盒式滤波器

图中显示了正方形的像素，其中心是由栅格线的交点表达的；权值函数表示成一个方块，其底就是当前的像素区域。根据图元覆盖像素的面积所决定的亮度是图元与该像素重叠区域的所有小块面积所决定的亮度的总和。每一小块面积所决定的亮度，是与其面积乘以权值的结果成比例的。所以，总的亮度是权值函数在重叠区域上的积分。由此积分W_S表示的体域，总是0和1之间的一个小数，而像素的亮度值I为$I_{\max} \cdot W_S$。在图3-37中，W_S是方块中的一个楔形。权值函数也称为**过滤函数**，而此处的方块亦称为**盒式滤波器**。对于未加权的区域采样，方块的高度都会规格化变成1，因此，方块的体积是1，这样，如果一条宽线盖住了整个像素，则该像素的亮度值$I = I_{\max} \cdot 1 = I_{\max}$。

现在，我们考察怎样为加权区域采样构造一个权值函数；它给远离像素中心的区域的权值一定要小于给靠近像素中心的区域的权值。我们在此挑选的权值函数是最简单的随着距离增大而递减的函数；例如，我们所选择的函数在像素中心有一个最大值，而随着远离中心的距离线性地递减。因为是中心对称的，这个函数的图形就形成了一个圆锥体。锥体的圆形基底（常被称为滤波器的**支集**）的半径应该比较大；滤波理论表明：对此半径的一种较好的选择是整数栅格的单位长度。这样，离像素中心相当远的图元对像素的亮度依然有影响；并且，相邻像素的支集也会有重叠，由此，即使图元的一小部分也可能会对几个不同的像素产生影响（见图3-38）。 123
这种重叠也确保了栅格上不会有不能被像素覆盖的区域。当圆形像素的半径只是栅格的单位距离的一半时，就可能会出现不能被像素覆盖的区域。⊖

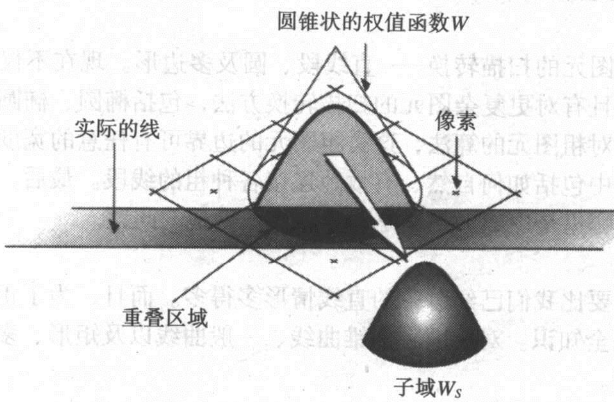


图3-38 针对圆形像素的锥形滤波器，其直径是两个栅格单位

同关于盒式滤波器的计算一样，为锥形滤波器计算的所有亮度的总和就是在锥顶的下方和锥体的

⊖ 在3.2节中已提到，在一个CRT上显示的像素的横截面大致是圆形的，并且相邻的像素一般会有重叠；但是，在加权区域采样中所用的有重叠的圆形模型与这种情况并不直接相关，甚至对显示技术而言也是这样，比如等离子体平板，其中物理像素实际上是没有重叠的正方形片。

基底与图元相交部分的上方之间的体域；体域W_S是锥体中垂直于基底的一个部分，如图3-38所示。像盒式滤波器中一样，锥体的高度首先被规格化，由此，整个锥体下的体积是1；这样，如果一个像素的支集被一个图元完全覆盖，则它可以以最大的亮度值显示。对于图元中远离像素的中心但仍与像素的支集相交的区域，尽管其对像素亮度的贡献很小，但如果像素的中心离线足够近，则它还是能从线获得一定的亮度贡献。相反，在像素被定义为方形几何的模型中，被一条有单位宽度⊖的线完全覆盖的方形像素，就不会显示得像它应该的那样亮。加权区域采样的作用其实就是降低相邻像素间的亮度对比，以便像素间的亮度可以平滑过渡。特别是，运用加权区域采样，单位宽度的水平线或垂直线在每一列或每一行中会有1个以上的像素具有亮度。这种情形，在未加权区域采样中不会出现。

124 锥形滤波器有两个有用的特点：中心对称，以及函数值随着半径的线性递减性。中心对称是很有用的，它不但使得有关区域的计算与线的倾斜角度无关，而且在理论上是最优的。注意，尽管锥形滤波器要优于盒式滤波器，但它的线性倾斜面（以及它的半径）只是对最优滤波器函数的一种逼近。最优的滤波器在计算上的开销是很大的，而方形滤波器的开销则是最少的，因此，在开销和质量两方面进行比较，锥形滤波器是一个比较好的在代价与质量间的折中选择。我们可以把锥形滤波器集成到我们的扫描转换算法中去，这一过程可以在[FOLE90]中找到。

3.15 高级主题

在本章中我们只是涉及了裁剪和扫描转换的概念。事实上，在高级方法的应用中会产生许多复杂的情况。这些将在[FOLE90]的第19章中进行完整的讨论，但在这里列举一些将是有用的。

1. 裁剪

在本章中已经讨论的裁剪算法，在多数情况下将能正确地执行，但它们并不总是高效的。此外，在某种情况下，它们即使给出了正确的答案，它们也是不精确的。改进后的一些算法，如2D裁剪的Nicholl-Lee-Nicholl方法显著地加速了梁友栋-Barsky 和Cohen-Sutherland两种算法。而且，裁剪中也可出现我们没有考虑过的情况，如用一般多边形裁剪其他的一般多边形。这种情形，Weiler多边形算法是可用的。

2. 扫描转换图元

我们只考虑了简单图元的扫描转换——直线段、圆及多边形。现在不仅有对这些图元更精确和更有效的算法，而且有对更复杂图元的扫描转换方法，包括椭圆、椭圆弧、三次曲线及一般的圆锥曲线。还有针对粗图元的算法，这类粗图元的边界可有任意的宽度，而不只是数学意义上的边界。这类问题中包括如何自然、有效地连接各种粗的线段。最后，在填充自相交的多边形时，必须常常考虑哪里是内部、哪里是外部这类不清楚的问题。

3. 反走样

需要反走样的情况要比我们已经讨论的直线情形多得多。而且，为了正确地反走样，我们必须有采样理论的更完全知识。对于圆、圆锥曲线、一般曲线以及矩形、多边形和线段端点等需要专门的反走样算法。

4. 正文

正文是一种非常专门的实体，我们较早的技术通常是不够充分的。在本章里，我们讨论了使用一种字体高速缓存来存储字符，于是该字符可直接复制到位图中，但是我们也看到此方法有一
125 定局限性：不同大小的每种正文字体需要一个不同的高速缓存，且字符的间隔是固定的。此外，

⊖ 我们现在是说“单位宽度的一条线”，而不是“单个像素宽的一条线”。这是为了明确地表达：在此线宽的一个单位依然是SRGP栅格的单位长，但像素的支集已变大，其直径有两个单位长。

黑体和斜体的正文版本虽然可以由高速缓存产生，但它们通常是不令人满意的。即使我们有（如可能是字体设计者提供的）一个字符的精确几何图画，我们也并不能按笔划来扫描转换它。其结果一般是不能被接受的。更合适地，已经开发了专门的技术来显示正文，包括相应的反走样。

5. 填充算法

当我们画了一系列图元后，有时我们希望给它们填上颜色，也可能我们希望在一个徒手作画的区域填上颜色。例如，用直线网格创建一个马赛克图案可能是容易的，然后用不同的颜色来填充它们，而不是用第一点处的颜色均匀地铺成有颜色的网格。请注意，当使用第一种技术时，并没有2D图元被绘制：我们在画了许多直线后，仅仅在2D区域内构建了一个背景。因此，要决定多大的区域被填上颜色需要检测边界什么时候到达。执行这个操作的算法称为*填充算法*。其中常用的有*边界填充*、*流入填充*、*浅色填充*等算法。每一种都有其特有的目的，许多图形系统提供全部算法。

小结

本章，我们先对基本的裁剪和扫描转换算法进行了讨论，它们是生成光栅图形软件包的基础。在此，我们只介绍了一些基本的内容；若要高效地实现程序，还必须考虑一些特殊情况，并做一些细致的改进。关于这些问题的更全面的介绍请参阅[FOLE90]的第14、17和19章。

内循环只进行整数运算的增量扫描转换算法，一般是最好的方法。这是本章最重要的思想，因为在交互式光栅图形学中，速度是最根本的要求。这些基本算法可以拓展来处理宽图元的情况，以及运用图案处理边界或填充区域的情况。对图元进行单个像素宽的扫描转换算法，都要尽可能地减少笛卡儿栅格上被选择的像素与定义在平面上实际的图元之间的误差，但那些处理宽图元的算法，基于速度的考虑，往往在质量方面和“正确性”方面要做一些牺牲。到目前为止，尽管许多二维的光栅图形学操作仍使用1位像素的图元，甚至在彩色显示器上也是这样，但我们希望实时反走样的技术能够很快地得到应用和普及。

习题

3.1 编写程序，以扫描转换水平线、垂直线和斜率为±1的线。

3.2 修改扫描转换线的中点算法（程序3-2），以便处理任意角度的线。

3.3 说明在扫描转换线的中点算法中，为什么点到线的误差总是小于等于1/2。

3.4 根据3.2.3节中所讨论的内容，修改习题3.2中的扫描转换线的中点算法，以处理端点排序以及与裁剪边相交的情况。 126

3.5 修改习题3.2中的扫描转换线的中点算法，使得写像素时的亮度是随着线的斜率而变化的。

3.6 修改习题3.2中的扫描转换线的中点算法，以处理不是整数坐标的端点的情况——如果在算法实现时，你用的全部是浮点数，这是非常容易做到的。但是，只用整数来处理端点是有理数的线就是一个比较难的问题。

3.7 说明在什么情况下折线可以不只共享顶点像素。对此，给出一个不必对像素写两遍的算法。提示：将扫描转换和以**xor**模式写画布的操作分解成独立的过程。

3.8 开发一种不同于3.3.2节中扫描转换圆的中点算法的算法，以便用折线来对圆进行分段的线性逼近并以此来扫描转换圆。

3.9 设计一个算法，以扫描转换空心的圆角矩形，其角都是给定半径的四分圆弧。

3.10 写一个扫描转换程序，可以在屏幕上的任意位置填实直立的矩形，并且能高效地写二值帧缓存，一次要写一个字的像素。

3.11 运用3.5节中的规则构造一些例子，使得一些像素被“遗失”或被写许多次。试提出另一套可能更复杂的规则，使得共享边上的共享像素不会画两次，也不会导致像素的漏画。这些规则与所增加的开销相比，是否值得？

3.12 实现3.5节中有关多边形扫描转换的伪代码，在跨度的记录中要考虑可能出现的狭长多边形。

3.13 利用三角形和梯形简单属性给出扫描转换它们的算法。在硬件中，这样的算法是很多的。

3.14 分析可以将任意的（可能是凹的或自交的）多边形分解成顶点共享的三角形网格的三角剖分方法。这是否有助于限制多边形的形状变化，使得多边形最坏就是凹的，但绝不会有自交情况和内部空洞情况的出现？（也可参见[PREP85]。）

3.15 运用跨度表对扫描转换圆的中点算法（程序3-4）进行扩展，以便能填充圆和圆状楔形（如圆饼状的图）。

3.16 对于多边形的图案填充，实现绝对定位和相对定位两个算法，这在3.7节中已讨论过。然后，比较它们在视觉效果和计算效率方面的差异。

3.17 运用图3-20中所示的技术，以不透明的方式对字符进行图案填充。对于这种问题的处理，讨论如何充分利用附带写掩码的copyPixel命令。

3.18 实现一种可以画各种符号的技术，比如用小型位图表示的光标图标，使得它们无论写到什么样的背景上都能看得见。提示：为每一种符号定义一个“包含”它的掩码（也就是，掩码要比符号能覆盖更多的像素），然后，分别画掩码和符号。

127 3.19 用3.7节中介绍的技术实现画宽线的算法。比较它们所得结果的质量和效率。

3.20 扩展扫描转换圆的中点算法（程序3-4），以处理宽的圆。

3.21 实现一个可以提供线型以及笔型和图案的画宽线的算法。

3.22 修改程序3-7中的Cohen-Sutherland线裁剪算法，以避免在连续的裁剪操作中反复计算斜率。并且，重新定义结构*outcode*为**unsigned int**（无符号整型数）all和四个标记位*left*, *right*, *bottom*, *top*的联合。

3.23 考虑一个有*n*个顶点的凸多边形被一个裁剪矩形裁剪的情况。裁剪后的多边形，顶点数最多是多少？最少是多少？对凹多边形，也考虑同样的问题。所得结果是多少个多边形？如果只得到一个多边形，它最大的顶点数可能是多少？

3.24 解释为什么Sutherland-Hodgman多边形裁剪算法只对凸裁剪区域有效。

3.25 设计一个方法将像素进行细分，并计数被线覆盖（至少要有比较明显的部分被覆盖）的
128 子像素个数，并以此作为采用未加权区域采样的画线算法的一部分。

第4章 图形硬件

本章将描述计算机图形显示系统中主要硬件的工作原理。4.1节覆盖了硬拷贝设备：打印机、笔式绘图仪、静电绘图仪、激光打印机、喷墨打印机、热传导绘图仪和胶片记录器。简要地叙述了每一类设备的基本技术概念，然后用一个结论小节比较了这些不同的设备。4.2节是有关显示设备的，讨论了单色和彩色荫罩式阴极射线管（CRT）、直视存储管（DVST）、液晶显示器（LCD）、电致发光显示器，又用一个结论小节讨论了不同显示设备的优点和缺点。

光栅显示系统可以使用这里讨论的任何一种显示技术，它将在4.3节讨论。我们先介绍一个简单直接的光栅系统，然后再讨论跟图形功能有关的方面以及光栅处理器和通用处理器如何集成到系统地址空间。4.4节讨论了在图像显示、颜色控制和图像合成中要用到的查找表和视频控制器的功能。接下来的4.5节讨论用户交互设备，如手写板、鼠标、触摸板等等。除了技术细节以外，对运作原理也进行了叙述。4.6节简要讨论了图像输入设备，如图像扫描仪，利用图像输入设备可以把现有的图像输入到计算机中去。

图4-1显示了这些硬件设备之间的关系。关键元素是集成的CPU和显示处理器，通常被称为**图形工作站**，一般由一个每秒至少执行2千万条至1亿条指令的CPU以及一个分辨率至少为1000×800（或更高分辨率）的显示器组成。局域网把多个工作站连接起来，这样可以共享文件、使用电子邮件， 129

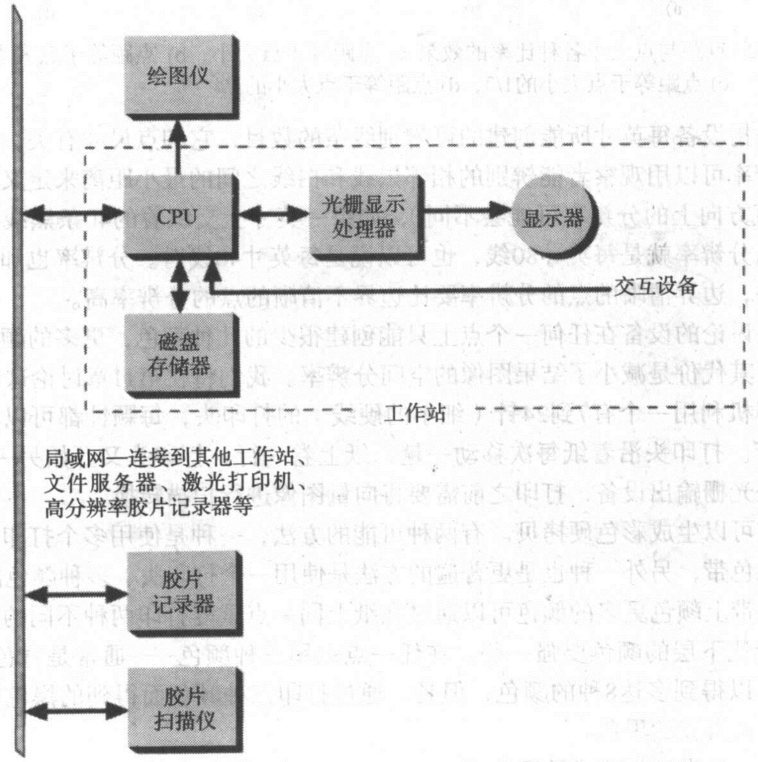


图4-1 典型交互式图形系统组成

并且可以使用其他的共享外设，如高品质的绘图仪、大硬盘还有连接其他网络的网关和更高性能的计算机。

4.1 硬拷贝技术

在这一节中我们将要讨论各种硬拷贝技术，然后概括它们的特点。但是，首先要定义几个重要的术语。

显示设备所能获得的图像品质与设备的寻址能力（addressability）以及点尺寸（dot size，也称为 spot size）有关。**点尺寸**是设备上创建的一个点的直径。**寻址能力**是指每英寸所能创建的点的数目（不要求可辨别），水平方向和垂直方向上的寻址能力可能不同。在x处的寻址能力是（x, y）坐标和（$x+1$, y）坐标中心点距离的倒数，在y处的寻址能力也类似定义。**点间距离**（interdot distance）是寻址能力的倒数。

130 我们总是希望点尺寸比点间距离大一些，这样可以获得平滑的形状。图4-2演示了其原理。这里进行了权衡：点尺寸是点间距离的几倍时，可以得到很平滑的打印形状，而较小的点尺寸则可以得到更好的细部特征。

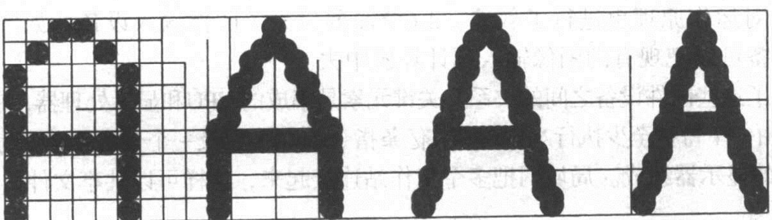


图4-2 点距与点大小各种比率的效果 a) 点距等于点大小，b) 点距等于点大小的1/2，c) 点距等于点大小的1/3，d) 点距等于点大小的1/4

分辨率是指设备每英寸所能创建的可辨别线条的数目，它和点尺寸有关，小于或者等于寻址能力。分辨率可以用观察者能辨别的相邻黑线和白线之间的最小距离来定义（这又意味着水平方向和垂直方向上的分辨率可能会不同），如果一英寸上交织着的40条黑线和40条白线能够被辨别，那么分辨率就是每英寸80线，也可以说是每英寸40线对。分辨率也和一个点的截面亮度的分布有关，边界清晰的点的分辨率要比边界不清晰的点的分辨率高。

很多将要讨论的设备在任何一个点上只能创建很少的几种颜色，更多的颜色可以通过抖动模式来得到，其代价是减小了结果图像的空间分辨率。我们将在第11章讨论这个问题。

点阵打印机利用一个有7到24针（细小的硬线）的打印头，每颗针都可以单独地触发，对着纸击打色带。打印头沿着纸每次移动一趟，纸上卷一行，打印头又开始另一趟打印，因此，这种打印机是光栅输出设备，打印之前需要将向量图像进行扫描转换。

彩色色带可以生成彩色硬拷贝。有两种可能的方法，一种是使用多个打印头，每个打印头使用一种颜色色带，另外一种也是更普遍的方法是使用一个打印头、多种颜色的色带。

比实际色带上颜色更多的颜色可以通过在纸上同一点重复打印两种不同的颜色来得到。在上层的颜色会比下层的颜色要强一些，在任一点上用三种颜色——通常是青色、紫色和黄色，重复打印后可以得到多达8种的颜色，但是，通过打印三种颜色而得到的黑色很模糊，所以经
131 常在色带上加入真正的黑色。

绘图仪的一种类型是**笔式绘图仪**（pen plotter），它可在一张纸上以随机的、向量的方式移

动笔，绘制直线的时候，笔定位在直线的起点，然后放下到纸上，沿着一条笔直的路径移向直线的终点，抬起，又移动到下一条直线的起点。

有两种基本的笔式绘图仪，桌面**平板绘图仪**在纸上沿*x*方向和*y*方向移动，纸是平铺在桌面上，由静电吸引力、真空吸引力或者就是简单地拉紧固定。一个机械车架在桌面上沿纵向移动，在车架上有一支笔沿车架横向移动，笔可以抬起和放下。平板绘图仪尺寸大小从12英寸×18英寸到6英尺×10英尺或者更大。

和上面形成对比的是**鼓式绘图仪**，鼓式绘图仪沿一个方向移动纸，沿另外一个方向移动笔。通常，纸是在鼓上拉紧的，鼓上的针扣着纸上预先打好的孔以防止纸滑动。鼓可以向前和向后转动。而许多**桌面绘图仪**可以在滚筒之间前后移动纸，笔则在纸上移动（图4-3）。

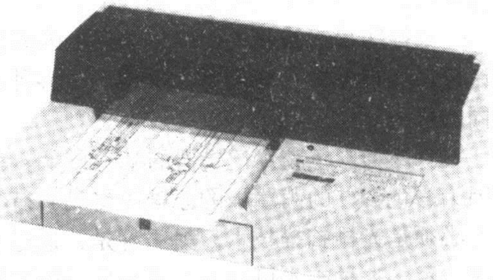


图4-3 桌面绘图仪（惠普公司提供）

激光打印机用一束激光扫描带正电的、其表面覆盖着一层硒的旋转鼓，被激光束击中的区域就失去正电，只有那些将要拷贝成黑色的地方才保留着正电，带负电的调色剂粘着到硒鼓上带正电的区域，然后转移到空白的纸上形成拷贝。在彩色激光打印机中，这个过程要重复三次，每次打印一种主色。图4-4是单色激光打印机的部分结构示意图。硒鼓上的任一个点要么带正电，要么不带电，在拷贝上相应的点要么就是黑色要么不是黑色，因此激光打印机是两级的单色设备或者八色的彩色设备。 132

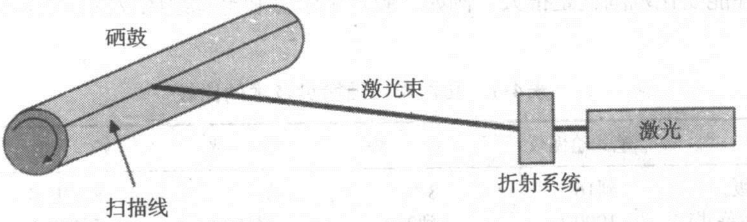


图4-4 激光打印机组成（色剂应用机制和送纸系统未表示）

激光打印机有一个微处理器来做扫描转换和控制打印机，越来越多的激光打印机把PostScript文档图像描述语言作为事实上的标准[ADOB85]。PostScript提供了要打印的图像的过程描述，PostScript也可以用于存储图像描述。大部分激光打印机使用8.5英寸×11英寸大小或者8.5英寸×14英寸大小的纸。但在工程绘制和地图绘制应用中也会用到相当宽（30英寸）的打印机。

喷墨打印机把青色、紫色、黄色，有时候还有黑色的墨水喷洒在纸上。在大多数情况下，墨水喷射器是安装在一个类似打印机结构的头上，打印头移动绘制一条扫描线，然后回车，纸往前走一个扫描线间距，接着绘制下一条扫描线。如果走纸太多或者太少，会做一些相应的细微调整。所有的颜色也都是同时沉积，而不是像多趟激光打印机和静电绘图仪那样分别沉积。多数喷墨打印机对每个像素的控制也只是限于开和关（就是二值）。一些则具有变点大小的功能。

热传导打印机是另外一种光栅硬拷贝设备，它利用均匀细密分布（通常是每英寸200颗）的加热笔尖把彩色蜡纸上的彩色粉剂传输到白纸上，加热笔尖带同时在蜡纸和白纸上绘制。笔尖有选择地被加热使得色剂能够传输。对彩色打印（这种技术最通常的用途），蜡纸是在一个交替着青色、紫色、黄色和黑色条带的卷轴上，每一条带的长度都和纸张的大小相等。因为笔尖的加热和冷却非常迅速，一张彩色图像的硬拷贝可以在一分钟内完成。一些热传导打印机能同时

接受视频信号输入和数字位图输入，这使得创建视频图像的硬拷贝非常方便。

热升华印染传导打印机和热传导打印机的工作原理类似，但是加热和印染传导过程可允许有256种不同强度的青色、紫色、黄色，可以创建出空间分辨率为每英寸200点的高品质全彩色图像，它的打印过程比蜡传输要慢，但是其质量可以和照片接近——为产生全彩色印前清样，采用这类打印机是一种明智的选择。

133 拍摄**阴极射线管**（CRT）上显示的图像的**照相机**可以看作是一种硬拷贝设备。这是我们所讨论的可以在单一分辨率点上得到许多种颜色的最普通的硬拷贝技术。胶片可以记录很多种不同的颜色。

有两种彩色胶片记录器的基本技术。一种是：照相机直接拍摄彩色CRT上显示的彩色图像，因为彩色CRT的荫罩板的原因（参见4.2节），图像的分辨率是受限制的，而且彩色显示器使用的必须是光栅扫描技术。另一种是：透过彩色滤镜拍摄黑白CRT，图像中不同颜色的元素按次序显示。这种技术可以得到很高质量的光栅或者向量图像。通过两个或者多个滤镜两次曝光图像的一部分以形成混合的颜色，通常所用的CRT的亮度也不同。

胶片记录器的输入可以是光栅视频信号、位图或向量指令。视频信号可以直接驱动彩色CRT，也可以将信号中的红绿蓝分量分离出来按时间顺序透过滤镜显示。这两种情况下整个摄制周期视频信号都必须保持恒定，如果使用的是较慢速度（低灵敏度）的胶片的话这个过程可能持续达1分钟。

表4-1总结了大部分彩色硬拷贝设备的不同之处。更多的关于硬拷贝设备技术的细节可以参阅[DURB88]。当然现在技术创新非常之迅速，某些设备的相对优点和缺点都会改变。某些设备其价格和性能变化范围也是很大。例如，胶片记录器和笔式绘图仪的价格从500美元到100 000美元都有。

表4-1　几种彩色硬拷贝技术的比较

	笔式绘图仪	点　阵	激　光	喷　墨	照　片
每点彩色等级	到16	8	8	8至更多	很多
寻址能力(点/英寸)	1000 +	到250	到1500	到200	到800
点大小(千分之一英寸)	15~6	18~10	5	20~8	20~6
相对价格范围	L~M	VL	M~H	L~M	M~H
每图像相对价格	L	VL	M	L	H
图像质量	L~M	L	H	M	M~H
速度	L	L~M	M	M	L

注：VL＝很低，L＝低，M＝中，H＝高。

注意，在所有这些彩色设备中，只有胶片记录器、热升华印染传导打印机以及一些喷墨打印机可以记录很宽范围的颜色，其他的技术都是对三种或者四种它们能够直接记录的颜色使用二元的开关控制。还要注意到这种颜色控制是需要技巧的，不能保证一个设备上的八种颜色和显示器上的一样，也不能保证它和另外一个硬拷贝设备上的一样有关颜色再现的固有困难的讨
134 论请参见[FOLE90]的13.4节。

4.2 显示技术

交互式计算机图形学需要可以迅速改变图像的显示设备。非永久性显示允许改变图像，使得动态移动图像的某部分成为可能，到目前为止，CRT是最普遍使用的显示设备而且还要普遍

使用很多年。但是，固态技术的发展从长久来看将会充分削弱CRT的统治地位。

用于图形显示的**单色CRT**和家用黑白电视机中所用的没有什么区别，图4-5显示了具有代表性的CRT剖面视图，由电子枪发出的电子流（阴极射线）受CRT封装内靠近射线管表面的高正电压的加速，射向涂覆荧（磷）光层的屏幕，在射向屏幕的过程中，电子束通过聚焦系统的控制汇聚成极细的一束，然后受偏转线圈产生的磁场控制射向屏幕上的特定位置，当电子击中屏幕时，荧光物质就会发射出可见光。因为荧光物质的发光输出随时间指数递减，整个图像必须每秒钟**刷新**（重绘）许多次，观测者看到的才是稳定的不闪烁的图像。

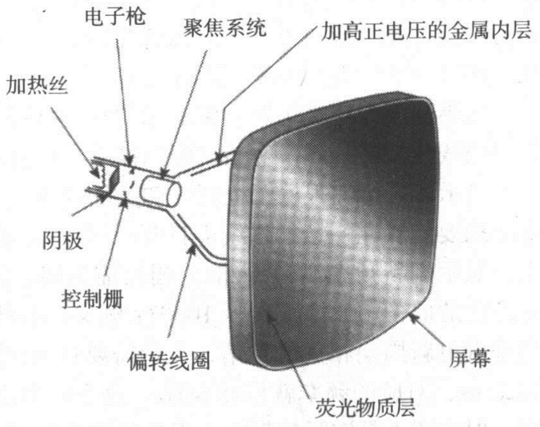


图4-5　CRT的剖面图（未按比例绘制）

光栅扫描显示系统的刷新速率和图像的复杂度是无关的。向量显示系统的刷新速率则直接和图像复杂度有关(直线、点和字符的数目)，图像复杂度越大，每个刷新周期所花的时间就越多，刷新速率就越低。 135

从加热的阴极射出的电子束被一个通常15 000伏到20 000伏的高电压加速，这个电压决定电子击中荧光物质之前的速度。控制栅电压决定电子束中实际有多少电子，这个负的控制栅电压越大，穿过栅的电子越少。这样就可以控制某个点的亮度，因为荧光物质的可见光随着电子束中电子数量的减少而减弱。

聚焦系统聚集电子束使得电子束击中荧光物质的时候会聚在一个小点上。仅仅让电子束中的电子平行移动是不够的，因为电子之间的相互排斥会使它们发散开，所以需要聚焦系统来汇聚它们以防止它们分散。除了电子束的这种分散的趋势以外，聚焦电子束和聚焦光线是类似的。

当电子束击中覆盖着一层荧光物质的CRT屏幕时，单个电子移动所带的动能和加速电压成正比，这个能量的一部分转化成热量，剩下的则传递给荧光物质原子中的电子，使它们跳跃到较高的量子能级，当这些电子恢复到它们原来的量子等级时，就会以光的形式释放出多余的能量，其频率（颜色）由量子理论可以预知。任何一种荧光物质都有几种不同的电子可以跳跃到的量子等级，从每个这样的等级返回未激活状态都对应着一种颜色。另外，电子在一些等级上要比在另一些等级上更不稳定，更加容易返回未激活状态。荧光物质的**荧光**（fluorescence）是指荧光物质受电子打击的时候，这些不稳定的电子丢失它们多余的能量从而释放出来的光线。**磷光**（phosphorescence）是指电子束移走之后，相对较稳定的激活电子返回到不激活状态时发出的光线。一般的荧光物质发出的大部分光线是磷光，因为激活和发生荧光只能持续几分之一微秒。荧光物质的**余辉**（persistence）是指从屏幕发光到磷光衰减到其原光强10%的时间。有些荧光物质的余辉可以多达数秒，但是对图形装置使用的大多数荧光物质而言，余辉为10μs到60μs。荧光物质的光线输出随时间指数递减，有关荧光物质特征的细节可参阅[SHER93]。

CRT的**刷新速率**定义为每秒钟里图像重绘的次数。光栅显示系统的刷新速率一般为每秒60次，当刷新速率减小的时候，因为人眼不能把从一个像素发出的单次光脉冲较长地整合在一起，就产生了**闪烁**（flicker）。当大于等于某个刷新速率的时候，闪烁停止，图像稳定，这个刷新速率就叫作**临界停闪频率**（critical fusion frequency，CFF）。停闪的过程对我们是熟悉的：当我们观看电视或者运动图像的时候就是这样，尽管实际上没有图像的时间比有图像的时间要长，

但无闪烁的图像对观看者看来是恒定的或者稳定的。

CFF的一个决定因子是荧光物质的余辉。余辉越长CFF越低，停闪频率和余辉之间的关系
136 是非线性的，余辉加倍并不能使得CFF减半。当余辉增加到几秒的时候，停闪频率变得非常小，而另一极端是，绝对没有余辉的荧光物质照样可以使用，因为所有的眼睛只要求在短时间里看见一些以大于CFF的频率重复的光。

水平扫描率定义为每秒钟驱动CRT的电路所能显示的扫描线的数目，它大概等于刷新速率和扫描线数目的乘积。对给定的扫描率，刷新速率增加意味着扫描线数目的减少。

显示器的**带宽**跟电子枪打开和关闭的速度有关，要达到每条扫描线n个像素的水平分辨率，电子枪要能够在一条扫描线上打开$n/2$次并关闭$n/2$次以获得交替开关的线，以1000线×1000像素，显示刷新频率为60 Hz的光栅扫描为例，简单计算表明，绘制一个像素需要的时间大约16 ns，它是量（1000像素/行×1000行/帧×60帧/秒）的倒数。事实上，因为有某些与每一个水平和垂直更新周期相关的开销，大约需要11 ns时间绘制一个像素。[WHIT84]。一个开关的周期为22 ns，对应的频率就是45 MHz，这个频率就是要获得1000条扫描线（500个线对）的最小带宽，但这还不是实际的带宽，因为我们忽视了点大小的影响。必须要用一个更高的频率来补偿非零的点的大小的影响，这样电子束打开和关闭会更快，从而使点的边界比没有补偿的要更清晰。一个1000线×1000像素的显示器其实际带宽达到100 MHz是很普通的。

彩色电视机和彩色光栅显示器利用的是某种形式的**荫罩式CRT**，在这种CRT中，显像管观察面的内侧表面上覆盖着紧密分布的多组红色、绿色、蓝色荧光点，这些点组非常的小，以至于从不同的点发射出来的光线被观察者看成是三种颜色的混合。因此，根据单个荧光点被激活的强弱不同，可以产生出很宽范围内的彩色来。荫罩是一块薄金属片，上面钻了很多小孔，它放置在与观测表面很接近的地方，并且经过很精密的对齐，这样三条电子束（一束击打红色荧光物质，一束绿色，一束蓝色）中的每条只能击打一种颜色类型的荧光点。因此，可以有选择地激活荧光点。

图4-6显示出了最普通的荫罩式CRT中的一种，即**delta-delta CRT**。荧光点以三角形**荧光点组**（triad）排列，三支电子枪也是以三角形排列，电子枪同时偏转，瞄准（会聚）的是观测面上的同一个点，荫罩上有一个小洞对应着每三个荧光点，这些洞与荧光点组和电子枪都经过精密细致的对齐，使得荧光点组中的每一个点只会暴露在一支电子枪发射出的电子下。高精度的delta-delta CRT的对齐特别困难。另一种可选的排列是按直线保证精度的**precision in-line delta CRT**，它更易于会聚和制造，是当前高精度（1000扫描行）监视器的一种可选工艺。可是，由于delta-delta CRT提供了较高分辨率，它很可能再度成为高清晰度电视（HDTV）的支配工艺。**平板彩色CRT**尽管还处于实验室研究阶段，但已经显现出了巨大的商业价值。在平板彩色CRT中，电子束平行于观测平面移动，然后再偏转90°击打荧光屏。

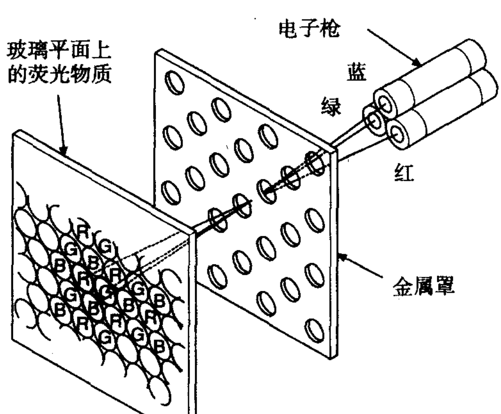


图4-6 delta-delta荫罩CRT。三支电子枪及荧光点按三角形(△)排列，荫罩使每支电子枪发射出的电子束只击中其对应的荧光点

对荫罩和荧光点组的需要使得彩色CRT的分辨率有了一个单色CRT所没有的限制。在非常高分辨率的显像管中，荧光点组分布在离三点中心0.21mm的地方，在家用彩色电视机的显像

管中分布在离中心0.60mm的地方（这个距离也叫作显像管的**点距**）。因为一束聚焦很好的电子束也不能保证能够正好击中荫罩的小洞的中心，所以电子束的直径（强度为最大强度的50%处的直径）必须是显像管点距的大约7/4倍。因此对一个点距为 0.25 mm（0.01英寸）的荫罩，电子束的直径大约为0.018英寸，而且分辨率不能高于1/0.018 = 55线每英寸。对一个点距为 0.25 mm、19英寸（沿对角线测量）（即大约15.5英寸宽、11.6英寸高[CONR85]）的显示器，能获得的分辨率仅为15.5 × 55 = 850乘以11.6 × 55 = 638。这个值可以拿来和通常的寻址能力1280 × 1024或者1024 × 800比较一下。如图4-2所示，比寻址能力略小的分辨率是有用的。

大多数高品质的荫罩式CRT对角线为15~21英寸，荧光屏略有弯曲，观测者会看到光学变形。几种类型的平面CRT逐渐变得实用。

液晶显示器（LCD）由六层组成，如图4-7所示。最表面一层是垂直偏振器，接下来的一层是在与液晶相邻表面上电沉积的垂直网格线层，再下面一层是一片薄（0.0005英寸）的液晶层，然后是与液晶层相邻的表面上有水平网格线的层，然后是水平偏振器，最后是一层反射器。 137~138

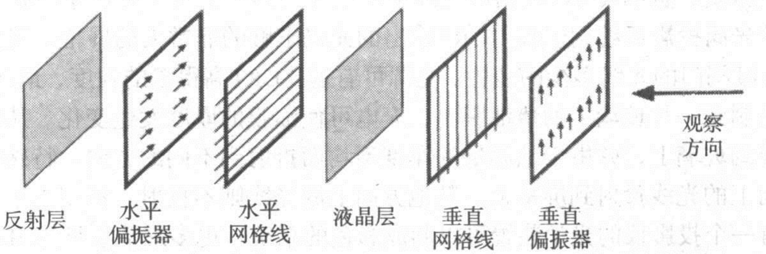


图4-7 液晶显示器的各层，所有层都拼贴在一起形成一薄板

液晶材料由长水晶分子构成，常态下分子排列成螺旋形，这样穿过的偏振光的偏振方向会旋转90°。进入前面一层的光线被垂直偏振，当光线穿过中间的液晶层时，光线的偏振方向被旋转90°成水平，这样光线就可以通过后面的水平偏振器，反射回来，再穿过两层偏振器和液晶。

当液晶处于电场中时，分子按照同一方向排成一线，就失去了偏振的作用，因此电场中的液晶不改变穿过光线的偏振方向，光线仍然保持垂直偏振，不能穿过后面的水平偏振器，光线被吸收，观测者在显示器上就会看到一个黑点。

一个在（x_1, y_1）上的黑点是通过*矩阵寻址*（matrix addressing）创建的，通过给水平网格线中的x_1加一个负电压 $-V$，给垂直网格线中的导线y_1加一个正电压 $+V$ 可以选择该点：单独的 $-V$和 $+V$还没有大到足够使液晶分子按照直线排列，但是 $-V$和 $+V$之间的电压差已经足够大，现在（x_1, y_1）处的液晶不再改变穿过光线的偏振方向，光线仍然保持着垂直偏振，不穿透后面的偏振器，光线被吸收，观测者在显示器上看到一个黑点。

有源阵列LCD面板（active matrix LCD panel）在每个（x, y）网格点放置一个晶体管，这些晶体管用来使液晶快速地转变状态，并用来控制液晶状态改变的程度，这两个属性使得液晶显示器可以用在具有连续色调的袖珍电视中。还能把液晶染上颜色以提供彩色，最重要的是，晶体管充当记录一个单元状态的存储器，并可以让这个单元一直保持这个状态直到被改变。也就是说，这个存储器让一个单元能够始终保持，因此要比需要周期刷新的单元亮度更高。已经 139
制造出了对角线14英寸、分辨率800 × 1000的彩色液晶显示器。

液晶显示器的优点是成本低、重量轻、尺寸小、能耗低。过去液晶显示器最主要的缺点是它是无源的，只是靠反射入射光而不自己发光（尽管可以用背后照明光改进）：强光会使得图像看不清楚。近年来，有源面板的应用使得这个缺点已经不再被考虑。事实上，带有彩色显示

器的膝上电脑（最近才可以使用）使用了有源LCD和无源LCD两种技术。因为LCD显示器又小又轻，还可以应用在头盔显示中，如8.1.6节中所讨论的。随着彩色显示屏的尺寸的增加及价格的不断下降，它们最终将威胁彩色CRT的主导地位——用不了几年。

电致发光显示器（electroluminescent display）有着与LCD和等离子平板类似的网格结构。在前后玻璃板中的是一薄层（通常为500 nm）电致发光物质，如涂有锰的锌的硫化物，电致发光物质在高电场（大约1 000 000 V/cm）下会发光，板上的一个点通过矩阵寻址的方案在水平和垂直选择线上加几百伏的电压来点亮。也有可用的彩色电致发光显示器。

这种显示器相当明亮而且可以快速地打开和关闭，每个像素位置的晶体管可用来存储图像。一般显示板的大小为6英寸 × 8英寸到12英寸 × 16英寸。每英寸上有70个可寻址的点，它的最大缺点是能耗比LCD的高，不过它们的明亮度使它们在一些便携式计算机中得以应用。

多数大屏幕显示器使用的是某种形式的**投射CRT**，在投射CRT中，从一个小（直径几个英寸）但是很明亮的单色CRT发射出的光线经过曲面镜放大并投射出来，彩色系统使用有红、绿、蓝三种滤镜的投影线。荫罩式CRT发出的光线不足以投射到一个大（对角线2 m）的屏幕上。

通用电气的**光阀投影系统**用于投射CRT输出的光仍不够的非常大的屏幕，光阀正如其名字所暗示：控制通过阀门的光线多少的装置。光源可有远比CRT高得多的强度，最普遍的方法中，电子枪把图像绘制到一片玻璃上的薄油膜上，充电可使油膜的厚度发生变化。从高强度光源发出的光线被引导到玻璃上，并由于油膜层的厚度不均而折射到不同的方向。特殊的光学装置把折射到特定方向上的光线投射到屏幕上，其他方向上的光线则不投射。利用三个投影仪或者使用更复杂的带有一个投影仪的光学装置可以制成彩色的系统。更多细节参见[SHER93]。

表4-2概括了三种主要显示技术的特点，然而，技术更新的步伐很快，在接下来的几年中一些关系可能发生改变。还要注意到，液晶显示的比较是对无源寻址的，使用有源阵列，可以
140 获得灰度等级和彩色。

更多关于这些显示技术的细节可参见[APT85；BALD85；CONR85；PERR85；SHER93；TANN85]。

表4-2 显示技术的比较

	阴极射线管	电致发光	液晶
能耗	中	中~良	优
屏幕尺寸	优	良	中
深度	差	优	优
重量	差	优	优
鲁棒性	中~良	良~优	优
亮度	优	优	中~良
寻址能力	良~优	良	中~良
对比度	良~优	良	中
每点亮度等级	优	中	中
视角	优	良	差
色彩能力	优	良	良
相对费用	低	中~高	低

4.3 光栅扫描显示系统

光栅图形系统的基本概念在第1章已经提及，第2章对光栅显示可能的操作类型进行了更深的探讨，在本节中我们讨论光栅显示器的各种元素，重点放在各种光栅系统相互不同的两个基

本方面。

第一，多数光栅显示器都有一个专门的硬件，用来帮助进行扫描转换，将输出图元转换成位图，并且执行移动、拷贝、修改像素或者像素块等光栅操作，我们把这个硬件称为**图形显示处理器**。显示系统之间最基本的不同在于显示处理器做多少工作，驱动光栅显示器的通用CPU上执行的图形子程序相对的又做多少工作。注意，有时候图形显示处理器也叫作**图形控制器**（强调它同其他外设的控制单元的相似性）或**显示协处理器**。第二个不同在于像素图与计算机通用内存的地址空间之间的关系，像素图是计算机通用内存的一部分，或者是独立的。

在4.3.1节我们介绍了一个简单的光栅显示器，它由一个CPU和一个视频控制器组成，像素图作为 CPU 内存的一部分，视频控制器驱动CRT。没有显示处理器，CPU既做了应用程序的工作也做了图形的工作。在4.3.2节，介绍了一个有单独像素图的图形处理器，4.3.3节讨论了广阔范围的图形处理器的功能，4.3.4节讨论了有图形处理器存在时，像素图可以集成到CPU地址空间中的方法。 141

4.3.1 简单的光栅显示系统

图4-8所示的是最简单和最普通的光栅显示系统的结构，内存和CPU之间的关系和其他非图形计算机系统的一样，但是，内存的一部分还充当像素图，视频控制器显示帧缓存中定义的图像，按照光栅扫描频率的规定通过一个独立的访问端口访问内存。在一些系统中，固定的一部分内存永久地分配给帧缓存，而一些系统有几个相同功能内存区域（在个人计算机中有时称为页），其他的系统则可以指定（通过寄存器）任意一部分内存作为帧缓存。

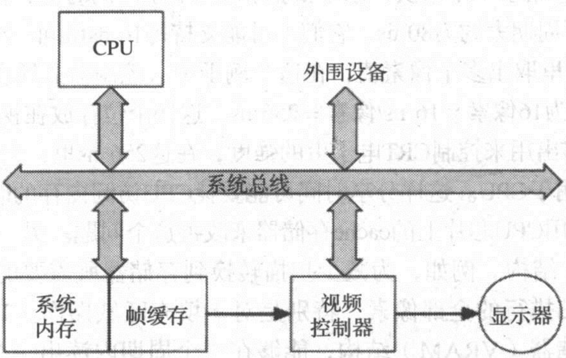


图4-8 普通光栅显示系统的结构，专用的一部分系统内存是双端口的，这样它可以直接被视频控制器访问，而不用中断系统总线的工作

应用程序和图形子程序包共享系统内存，并由CPU执行。图形软件包包含扫描转换过程，当应用程序调用子程序时，比如说调用SRGP_lineCoord（$x1, y1, x2, y2$），图形软件包能设置帧缓存中适当的像素（关于扫描转换过程的细节参见第3章）。因为帧缓存在CPU的地址空间里，图形软件包可以很容易地访问它来设置像素，并实现第2章里描述过的PixBlt指令。

视频控制器在帧缓存里轮转，每次一条扫描线，通常是每秒钟60次，内存引用地址和光栅扫描同步生成，内存中的内容用来控制CRT电子束的亮度或者颜色。视频控制器的结构如图4-9所示。光栅扫描生成器产生偏转信号，这些信号用来产生光栅扫描，它还控制X地址寄存器和Y地址寄存器，这两个寄存器依次定义了下一个要访问的内存位置。 142

假设帧缓存的x编址是从0到x_{max}，y编址是从0到y_{max}，那么在一个刷新周期的开始，X地址寄存器设置为0，Y地址寄存器设置为y_{max}（最顶部的扫描线），当第一条扫描线生成的时候，X地址每次增加1直到 x_{max}，每个像素的值被取出，并用来控制CRT电子束的强度，第一条扫

描线生成完毕后，X地址寄存器复位为0，Y地址寄存器减1，然后依次处理该条扫描线上的各个像素，整个过程对后面的扫描线重复执行，直到最后一条扫描线（$y=0$）生成。

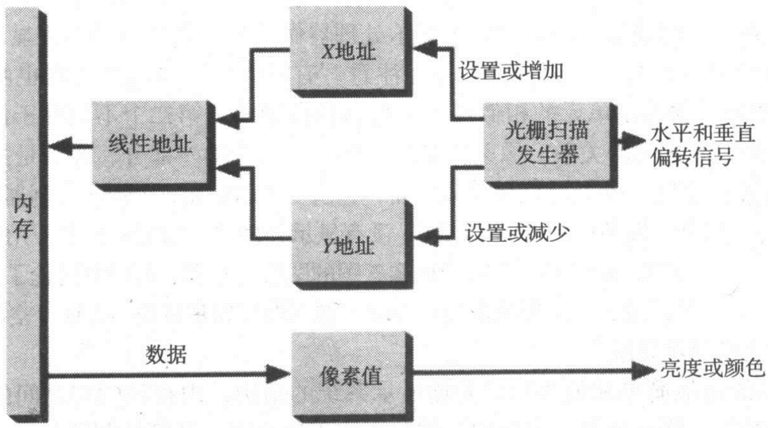


图4-9 视频控制器的逻辑结构

在这种简单的情形下，对每一个要显示的像素的帧缓存都要进行一次内存访问，对一个1000像素×1000行的高分辨率显示器，估计显示一个1位像素所用时间的简单方法是：1/(1000×1000×60) = 16 ns。这里忽略了一个事实：水平回扫和垂直回扫⊖的时候，像素都不显示，但是一般的RAM存储器芯片循环周期大约为80 ns，它们不可能支持每16 ns访问一次的要求！因此视频控制器必须在一个内存周期里取出多个像素值，在这个例子中，控制器必须在一个内存周期里取出16个位，从而刷新时间变为16像素×16 ns/像素 = 256 ns。这16个位存放在视频控制器的寄存器里，
143 然后每16 ns一个位逐位移出用来控制CRT电子束的强度，在这256 ns里，大约有3个内存周期：一个用于视频控制器，2个用于CPU。这样分享时间可能要让CPU访问内存的时候等待，就有可能降低CPU速度。当然可以使用CPU芯片上的cache存储器来改善这个问题。另一种方法是对于帧缓存使用非传统的存储器芯片结构。例如，为减少扫描转换到存储器所需要的存储周期数目，在一次存取时间内开启一个扫描行的全部像素，特别是对于填充区域操作。Texas Instruments公司开发的视频随机存取存储器（VRAM）结构，能够在一个周期内读出一个扫描行的全部像素，因而减少了刷新显示器所需要的存储周期的数目。

迄今为止我们讨论的只是单色的，每个像素一个位的位图。这个假设对一些应用是可以的，但对其他应用却非常的不满意。对每个像素的亮度的附加控制可以通过为每个像素存储多个位来获得：2位可以获得4个亮度等级，等等。这些位不仅可以用于控制亮度，还可以用于控制颜色。需要多少个位才能使存储的图像看上去是具有连续灰度的？通常5或者6位就足够了，但是8位或者更多位可能是需要的。因此，对彩色显示器，一个有些简单的说法提出需要三倍的位数：添加的原色红、绿、蓝中的每种需要8个位。

尽管每像素24位的系统相对不算昂贵，但许多彩色应用在一幅图像中不需要2^{24}种不同的颜色（一般只有2^{18}~2^{20}种）。而且，在一幅给定的图像或者一个应用中经常需要使用很少的颜色，也需要从图像到图像或者从应用到应用之间改变颜色的能力，还有，在许多图像分析和图像增强的应用中，想要改变图像的视觉效果，但是不改变定义图像的基本数据。比如说，把所有值

⊖ 在没有图像被跟踪期间，在光栅扫描系统中还要花费一定时间：每一扫描行出现一次的水平回扫时间和每一帧出现一次的垂直回扫时间。

低于某个阈值的像素显示成黑色，把亮度范围扩大，给单色图像创建出伪彩色显示。

由于这些各种各样的原因，视频控制器经常包括一个**视频查找表**（video look-up table）（也称为**查找表**(look-up table, LUT)），查找表的条目数和像素值一样多，像素值不再用来直接控制电子束强度，而是作为索引值来访问查找表，查找表中条目的值再用来控制CRT的亮度或者颜色。一个值为67的像素就表示访问查找表中第67项的内容，并且用这个内容控制电子束强度。在一个显示周期中对每个像素进行这样的查找操作，所以查找表必须能够快速访问，CPU必须能在程序命令的时候装入查找表。

在图4-10中，查找表插入在帧缓存和CPU的中间，帧缓存每个像素有8位，所以查找表有256项。

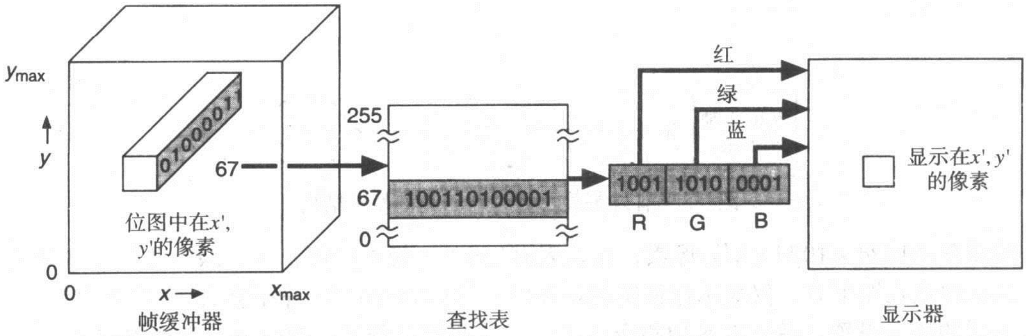


图4-10 视频查找表的结构。值为67（二进制值01000011）的像素显示到屏幕上，红色电子枪的强度为最大值的9/15，绿色的为10/15，蓝色的为1/15，所示的是12位的查找表，通常的会达到24位

图4-8显示的简单光栅显示系统的结构使用在很多便宜的个人计算机中。这样的系统造价低，但是有很多缺点。首先，软件扫描转换很慢，例如一条扫描线上的每个像素的（x, y）地址都要计算，然后转换成由一个字节和字节中的位组成的内存地址。尽管每一步都很简单，但是都要重复许多次，基于软件的扫描转换使得应用程序与用户交互的速度全部变慢，可能会使用户不满。

这种结构的第二个缺点是：当寻址能力或者显示刷新速率增加时，视频控制器的内存访问次数也随之增加，因此降低了CPU可用的内存周期的数目，CPU也因此慢下来。特别是对于必须通过系统总线访问帧缓存的结构。在图4-8中，系统内存有一部分是双端口的，当CPU访问帧缓存以做扫描转换或者光栅操作时，速度下降就会发生。在考虑CPU访问帧缓存的方便性和系统的结构简单性时，也还要考虑这两个缺点。

4.3.2 具有外围显示处理器的光栅显示系统

具有外围显示处理器的光栅显示系统是一种避免了简单光栅显示器的缺点的普遍结构（见图4-11），它引入了一个独立的执行诸如扫描转换和光栅操作等图形功能的图形处理器和一个独立的用于图像刷新的帧缓存。现在我们有两个处理器：通用CPU和专用显示处理器。我们还有三块内存区域：系统内存，显示处理器内存和帧缓存。系统内存存放数据以及在CPU上执行的程序：应用程序，图形软件包和操作系统。相似地，图形处理器内存上也存放着数据和进行扫描转换及光栅操作的程序。帧缓存存放扫描转换和光栅操作生成的可显示的图像数据。 144~145

在简单的情况下，显示处理器可以包含特定的逻辑，来完成从二维（x, y）坐标到线性内存地址的转换。在这种情况下，扫描转换和光栅操作仍由CPU完成，所以显示处理器内存是不需要的，只需要帧缓存。多数外围显示处理器还完成扫描转换，在本节中，我们讨论一个原型系统，它包含了许多一般商用系统的特征（有些经过简化），如使用在IBM PC兼容机中的插入式图形卡。

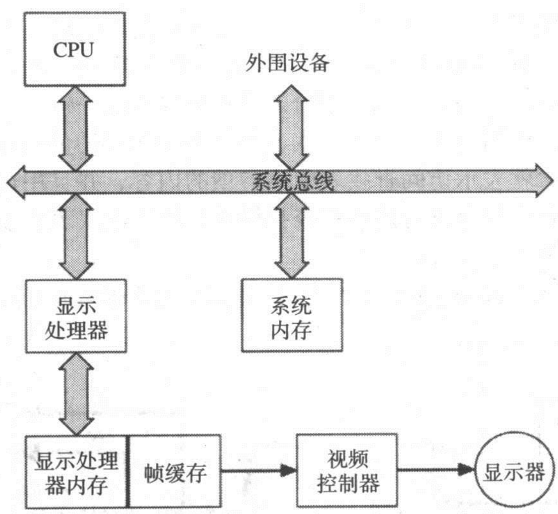


图4-11 有外围显示处理器的光栅显示结构

帧缓存是1024×1024×8位/像素，查找表有256项，每项12位，红、绿、蓝每种颜色使用4位，坐标原点在左下方，仅显示像素图的前768行（*y*从0到767），显示器有6个状态寄存器，可以被不同的指令设置，并影响其他指令的执行。这些寄存器是：CP（由X位置寄存器和Y位置寄存器组成），FILL，INDEX，WMODE，MASK和PATTERN。下面将解释它们的操作。

简单光栅显示的一些指令如下：

- Move（*x*, *y*） 将定义当前位置（CP）的X、Y寄存器设置成 *x* 和 *y*，因为像素图是1024×1024的，*x*和*y*必须在0到1023之间。
- 146 MoveR（*dx*, *dy*） *dx*和*dy*的值被加到X、Y寄存器，因此定义了新的CP，*dx*和*dy*必须在−1024到+1023之间，以2的补码表示。所做的加法可能导致溢出，因此X寄存器和Y寄存器环绕式处理。
- Line（*x*, *y*） 从CP到（*x*, *y*）画一条直线，（*x*, *y*）成为新的CP。
- LineR（*dx*, *dy*） 从CP到CP+（*dx*, *dy*）画一条直线，CP+（*dx*, *dy*）成为新的CP。
- Point（*x*, *y*） 设置（*x*, *y*）处的像素，（*x*, *y*）成为新的CP。
- PointR（*dx*, *dy*） 设置CP+（*dx*, *dy*）处的像素，CP+（*dx*, *dy*）成为新的CP。
- Rect（*x*, *y*） 在CP和（*x*, *y*）之间画一个矩形，CP不受影响。
- RectR（*dx*, *dy*） 在CP和CP+（*dx*, *dy*）之间画一个矩形，参数*dx*可以看成是矩形的宽，*dy*看成是矩形的高，CP不受影响。
- Text（*n*, *address*） 从CP开始显示内存位置为*address*的*n*个字符。字符定义在7×9的像素网格中，垂直和水平分别有额外的2个分隔像素用于分隔字符和行。CP更新为第*n*+1个字符将被显示的区域的左下角。
- Circle（*radius*） 以CP为圆心画一个圆，CP不受影响。
- Polygon（*n*, *address*） 地址为*address*的内存存储顶点列表（$x_1, y_1, x_2, y_2, x_3, y_3, \cdots, x_n, y_n$），以（$x_1, y_1$）为起点画多边形，经过所有这些顶点直到（$x_n, y_n$），然后回到（$x_1, y_1$），CP不受影响。
- AreaFill（*flag*） *flag*用来设置光栅显示中的FILL标志，当此标志设置成ON的时候（用一个非零的*flag*值），用命令Rect、RectR、Circle、CircleSector、Polygon创建的区域都会

用Pattern命令定义的图案填充。

- RasterOP（*dx, dy, xdest, ydest*）　将帧缓存中从CP到CP+（*dx, dy*）的矩形区域和左下角为（*xdest, ydest*）同样大小的目标区域结合，目标区域被覆写，结合受WMODE寄存器的控制。

命令和立即数通过位于CPU地址空间中专用部分的**先进先出**（FIFO）缓冲区（如队列）传输到显示处理器，图形软件包把命令送入到队列中，显示处理器取得指令并执行。在特定的内存位置还存放着指向这个缓冲区的起始地址和结束地址的指针，供CPU和显示处理器访问。每次移走一个字节时显示处理器修改指向起始地址的指针，每次加入一个字节时CPU修改指向结束地址的指针，要做适当的测试以保证对空的缓冲区不进行读操作，对满的缓冲区不进行写入操作，使用直接内存访问来给指令提供寻址数据。

对于命令传递，队列比使用单个指令寄存器或显示处理器可以访问的位置要更吸引人。第一，变长的指令适合队列的概念；第二，CPU可以超前于显示处理器，把许多显示命令排在队 147
列里。当CPU发布完显示命令后，在显示处理器处理命令清空队列时它可以去处理其他工作，

对显示的编程有点类似于第2章所述的SRGP软件包的使用。在[FOLE90]的第4章中介绍了几个编程示例。

4.3.3 显示处理器的附加功能

我们的简单显示处理器只完成一些能被实现的与图形相关的操作。设计者所面临的诱惑是：给显示处理器增加功能，以更多地减轻CPU的负担，比如使用局部内存来存放显示指令列表，完成裁剪和窗口-视口转换，也许还提供拾取相关逻辑和图形元素被拾取后的自动反馈。最终，显示处理器变成了另一个完成通常图形交互工作的通用CPU，设计者又要尝试着增加特定功能的硬件以减轻显示处理器的负担。

Myer和Sutherland在1968年确定了这个**轮回**（wheel of reincarnation）[MYER68]。作者的观点是在通用功能和专用功能之间要有一个折衷，通常，专用硬件完成工作比通用处理器快。另一方面，专用硬件更昂贵且不能用于其他用途。这种折衷在图形系统设计中是持久的题目。

如果裁剪（第3章）增加到显示处理器的功能中，输出图元可以用坐标指定给处理器而不使用设备坐标。这种说明可以在浮点坐标中给定，虽然一些显示处理器只能处理整数（随着低廉的浮点芯片的使用，这种情况正在改变中）。如果只使用整数，应用程序使用的坐标也必须是整数的，或者图形软件包必须把浮点坐标映射到整数坐标。要使映射成为可能，应用程序必须给图形包一个矩形，这个矩形保证包含了指定给图形包的所有输出图元的坐标，然后这个矩形映射到最大的整数范围，这样在这个矩形内的一切都是在整数坐标的范围里的。

如果子程序包是三维的，显示处理器就能够完成第5章和第6章中所述的复杂得多的三维几何变换和裁剪。同样，如果图形包里包含了三维曲面图元，比如多边形区域，显示处理器还能进行第13章和第14章讨论的可见面判定（visible surface-determination）和绘制步骤（rendering step）。在[FOLE90]的第18章中讨论了一些能使这些步骤完成得更快的把通用和专用VLSI芯片组织在一起的基本方法。许多商用的显示器都提供了这些特征。

另外一个经常给显示处理器增加的功能是本地段存储（local segment storage），也叫**显示列表存储**（display list storage）。显示指令被分组到命名的段中，具有未被裁剪的整数坐标， 148
并存储在显示处理器内存中，允许显示处理器的操作更自主于CPU。

显示处理器对这些存储起来的段能做些什么呢？它可以对它们进行变换和重绘，像缩放和滚动。能够提供这些段到新位置的本地拖动功能。通过让显示处理器对光标位置和所有图形图元

（更有效率的方法在第7章讨论）比较来实现本地拾取。当删除一个段的时候，需要用段存储来重新生成以填充产生的洞。段可以被创建、删除、编辑，还可以使段可见或者不可见。

段还可以被拷贝或者引用，这两种操作都减少了必须从CPU传送给显示处理器的信息，也使显示处理器自己的内存使用更经济。利用这个功能可以建立复杂的层次数据结构，许多具有本地段内存的商用的系统可以拷贝和引用其他段。当显示段时，必须保存显示处理器的当前状态，然后进行另一个段的引用，就像保存CPU的当前状态，然后进行子程序调用一样。引用可以嵌套，导致了**结构显示文件**或**层次显示列表**的出现，如在PHIGS中 [ANSI88]，第7章将进行更深的讨论。

尽管和4.3.1节中的简单光栅显示系统相比，具有图形显示处理器和独立帧缓存的光栅显示系统结构有着许多优点，它也还是有着一些缺点。如果显示处理器是作为DMA端口上或者RS232接口上的外设，则每次传送指令给它的时候就会有相当多的操作系统管理开销（显示处理器的指令寄存器映射到CPU的地址空间就不会发生这种情况，因为图形包很容易就可以直接设置寄存器）。

光栅操作命令特别复杂，从概念上说，它应该有四种可能的源-目的对：系统内存到系统内存，系统内存到帧缓存，帧缓存到系统内存，帧缓存到帧缓存（在此，图4-11中的帧缓存和显示处理器的内存被认为是相同的，因为它们都在同一个系统地址空间里）。但是在显示处理器系统里，对不同的源-目的对使用不同的方式处理，可能系统内存到系统内存的这种情况不存在。缺乏对称性使得程序员的任务变得复杂，降低了灵活性。比如说，如果位图超出屏幕的部分填充着的是菜单、字体等，就很难利用主存来作为溢出区域。更进一步，因为像素图的使用是如此的广泛，不支持对存储在主存中的像素图进行光栅操作是不可行的。

本节前面定义的显示处理器像许多真正的显示处理器一样，通过系统总线上的I/O传输在系统内存和帧缓存之间移动光栅图像。但是，实时操作里这种移动可能太慢，像动画、拖动、
149 弹出窗口和菜单，操作系统初始化传输的时间和总线的传输率是瓶颈所在。通过增加显示处理器的内存以装载更多的屏外像素图可以部分克服这个问题，但那样一来这部分内存就不能用于其他用途了——总之，几乎永远都不会有足够多的内存！

4.3.4 具有集成显示处理器的光栅显示系统

通过把系统内存的特定部分专用于帧缓存和通过提供从视频控制器到帧缓存的第二个访问端口，我们可以克服上节所讨论的外围显示处理器的许多缺点，这就是图4-12所示的**单地址空间**（single-address-space, SAS）显示系统体系结构。这里显示处理器、CPU和视频控制器都在系统总线上，因此都可以访问系统内存。帧缓存的起始地址，一些情形下还有大小，都存放在寄存器中，双缓存就变得很简单，只需重新填入寄存器：扫描转换的结果可以送到帧缓存用于直接显示，或者送到系统内存别的地方用于以后的显示。类似地，显示处理器进行的光栅操作的源和目的可以在系统内存的任何地方（现在我们只对内存感兴趣）。这种安排还有一点吸引人之处，是因为CPU可以直接操作帧缓存中的像素，只要简单地读写合适的位就可以。

但是，SAS体系结构有许多缺点，对系统内存访问的竞争是最严重的。一个解决方法是使用带有指令或数据高速缓冲存储器的CPU，减少CPU对频繁快速访问系统内存的依赖。当然这些方法和其他方法可以巧妙地集成到一起，[FOLE90]的第18章将讨论更多的有关细节。

如果CPU具有虚拟地址空间时，出现了另一个设计上的复杂因素，像普遍应用的Motorola 680x0和Intel 80x86系列，以及各种各样的精简指令集计算机（RISC）处理器。在这种情况下，显示处理器生成的内存地址和其他内存地址一样要进行同样的动态内存地址转换。另外，许多

CPU结构区分核心操作系统虚拟地址空间和应用程序虚拟地址空间，经常也需要帧缓存（在SRGP术语中是画布 0) 处于核心空间，这样操作系统的显示设备驱动程序可以直接访问。但是应用程序所分配的画布必须处在应用程序空间。因此访问帧缓存的显示指令必须区分核心地址空间和用户地址空间。如果访问的是核心，就应该由耗时的操作系统服务调用来调用显示指令，而不是由简单子程序调用。

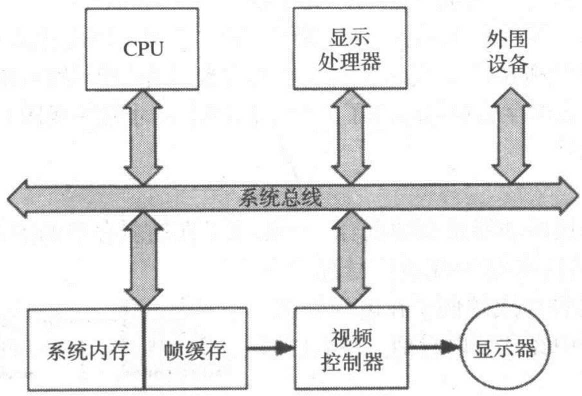


图4-12 公用单地址空间（SAS）光栅显示系统的体系结构，具有一个集成的显示处理器，该显示处理器可以有私有的用于存放算法的内存和工作存储器，系统内存的专用部分是双端口的，以便视频控制器可以直接访问，而不用中断系统总线的工作

尽管有这些潜在的复杂性，实际上越来越多的光栅显示系统使用单地址空间结构，一般是图4-12所示的那种类型。允许CPU和显示处理器用统一的方式访问内存的任何部分这一灵活性非常引人注目，并且编程简单。

4.4 视频控制器

视频控制器的最主要任务是持续地刷新显示。有两种基本的刷新方式：**交错的和不交错的**。前者使用在广播电视和设计用来驱动正规电视的光栅显示中。其刷新周期被分成两个部分，每部分持续1/60秒，一次完整的刷新持续1/30秒，所有奇数行扫描线在第一部分时间里显示，所有偶数行扫描线在第二部分时间里显示。隔行（交错）扫描的目的是每次以60 Hz的频率在屏幕的所有区域放置一些新的信息，因为30 Hz的刷新频率容易导致闪烁。交错显示的净效果是产生的图像其有效刷新频率更接近60 Hz而不是30 Hz。在相邻扫描线实际上显示相似信息时这一技术较为有用；在交替的扫描线上有水平线的图像将有严重的闪烁。大部分视频控制器以60 Hz或者更高的刷新频率刷新并且使用非交错扫描。

视频控制器的输出分为以下三种模式：RGB、单色和NTSC。对RGB（red，green，blue），用单独的电缆来传输红、绿、蓝信号，控制荫罩式显示器的三支电子枪，另外一根电缆传输标志开始垂直回扫和水平回扫的同步信号。RGB信号的电压、波形和同步时间都有标准，对480扫描线的单色信号，RS-170是其标准；彩色信号的标准是RS-170A；高分辨率单色信号的标准是RS-343。同步时间经常和绿色信号包含在同一根电缆里，这种情况下信号叫作复合视频（composite video）信号。单色信号使用同样的标准，但是只有亮度和同步信息，或者仅仅只是一根传输亮度和同步信号的复合电缆。 150~151

NTSC（国家电视系统委员会）视频是北美电视商品中使用的标准。颜色、亮度和同步信息都合并在一个带宽为5 MHz的信号中，以525条扫描线进行广播，分为两个262.5条扫描线的

部分，只有480条扫描线是可见的，剩下的扫描线在每个部分结束进行垂直回扫的时候出现。单色电视使用亮度和同步信息，彩色电视还使用彩色信息控制三支电子枪。带宽的限制可以允许在分配给电视的频率范围内使用许多不同的频道进行广播。不幸的是，这个带宽限制了图像的质量，使得它的有效分辨率只有350×350。不过，NTSC是录影带录制装置的标准。价格更贵的刻录机用模拟或数字的形式分开存储颜色信号的各个分量。欧洲和俄罗斯电视广播和录影带标准是SECAM和PAL，两个625扫描线、50 Hz标准。

一些视频控制器加入了可编程光标，光标的形状存储在16×16或者32×32大小、位于帧缓存顶部的像素图中，这样就避免了每次刷新周期里都需要把光标形状PixBlt到帧缓存中去。类似地，一些视频控制器在帧缓存的顶部增加了几个小的固定尺寸的像素图（叫作精灵（sprite）），这个特征经常用于视频游戏。

视频混合

视频控制器另一个有用的功能是视频混合。一幅定义在帧缓存中的图像可以和一个来自电视摄像头、录像机或者其他来源的视频信号混合一起形成一个复合图像。这种合成的例子在电视新闻、体育运动节目和天气预报中经常可以看到，图4-13描绘了通常的系统结构。

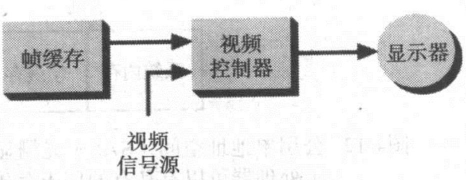


图4-13 把帧缓存的图像和视频信号源图像合成到一起的视频控制器

有两种类型的合成，一种是把图形图像置到视频图像中去，新闻报告员肩上显示的图表或者图形就是这个类型的典型例子，这种合成可以由硬件实
152 现，它把帧缓存中指定的像素值作为一个标志，指示应该显示视频信号，还是显示帧缓存中的信号，通常这个指定的像素值和帧缓存图像的背景颜色相对应，但是使用其他像素值能够获得感兴趣的效果。

第二种合成是把视频图像放置在帧缓存图像的上面，就像天气预报员站在全屏幕的天气图前面那样，天气预报员实际上是站在一个背景前，背景的颜色（通常为蓝色）用来控制合成：当视频信号是蓝色时，显示帧缓存图像，否则显示视频图像，只要天气预报员不穿蓝色衬衫或者系蓝色领带，这种技术会很好地工作。

4.5 用于操作者交互的输入设备

在本节中我们描述最普通的输入设备的工作。我们简单并高层次地讨论可用的设备是如何工作的。在第8章我们将讨论各种输入设备的优缺点，还描述一些更先进的设备。

我们的介绍是围绕着**逻辑设备**（logical device）的概念进行组织的，逻辑设备在第2章中已有介绍，并在第7章中进行更深入的讨论。有五种基本逻辑设备：**定位设备**，用来指明位置或者方向；**拾取设备**，用来选择显示的实体；**定值设备**，用来输入一个实数；**键盘**，用来输入字符串；**选择设备**，用来选择可能的行为或者选项集合中的一个或者多个。根据设备提供给应用程序的信息种类，逻辑设备的概念定义了这些输入设备的等价分类。

4.5.1 定位设备

1. 输入板

输入板（或**数据输入板**）是一个平板，其尺寸从6英寸×6英寸到48英寸×72英寸或者更大。它可以探测用户手中可移动触笔或者手持光标定位器的位置。图4-14显示了一个同时具有触笔和光标定位器（我们今后将主要只提及触笔，尽管讨论对两者相关）的小输入板。大部分输入板使用一种电感应的机制来确定触笔的位置，在一种设计中，网格宽度为1/4英寸或者1/2英寸

的矩形网格线嵌在输入板的表面，沿金属线生成电磁脉冲系列，激励触笔中的线圈，感应出电信号，每个脉冲感应出来的电信号的强度可以用来确定触笔的位置。这个信号强度还粗略地用于确定触笔或者光标距离输入板有多远（“远”、“近”（比如说离数据板大约1/2英寸）或者“触及”）。如果答案是“近”或者“触及”，显示器上显示出光标，给用户提供一个反馈，当触笔尖端在输入板上按压，或者手持光标定位器上的任一个按钮被按下时，会传送给计算机一个信号。每秒钟内可以30~60次获取输入板的（x, y）位置、按钮状态以及靠近程度状态（如果靠近程度状态为“远”时，没有（x, y）位置可用）。

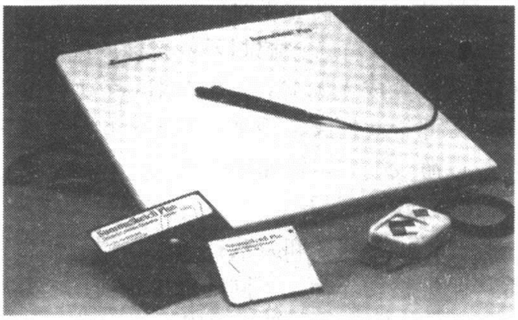


图4-14 带有触笔及光标定位器的数据输入板。触笔顶端有一个压敏开关，当触笔下压时，开关闭合。光标定位器有几个按键，用于输入命令，还有一个十字准线光标，用于精确地将输入板上的图形数字化后输入计算机。（Summagraphics公司提供。）

与输入板或者其他定位设备相关的参数有分辨率（每英寸可区别的点的数目）、线性度、可重复性和尺寸或范围。这些参数对于把地图或者工程图数字化特别重要，当设备只用于定位屏幕光标时就没有那么多考虑了，因为用户可以通过显示器上的反馈来指导他的手的移动，并且一般显示器的分辨率要比便宜的输入板的分辨率低得多。其他的输入板技术使用声音耦合或者电阻耦合。

有几种输入板是透明的，在数字化X射线片和照片底片时可以用背光照亮，也可以直接安装在CRT的前面，这时用电阻输入板特别适合，因为它可以按照CRT的形状弯曲。

2. 鼠标

鼠标是小型的手持设备，它在平面上的相对移动可以测量出来，鼠标之间的不同之处在于鼠标按钮的数量和测量相对运动的方法。各类鼠标之间的许多重要区别在8.1节中讨论。**机械鼠标**底部滚轮的移动被转换成用于确定移动的方向和数量的数字值。**光学鼠标**在专门的鼠标衬板上移动，衬板上交替着亮线和暗线的网格，鼠标底部的发光二极管把光线直射到衬板上，光线反射回来被鼠标底部的探测器所感应。当鼠标移动时，每次穿过暗线，反射光线都会被中断，产生的脉冲数量和穿过的暗线数量相等，用来向计算机报告鼠标的移动。

因为鼠标是相对设备，它可以拿起来，移动，然后再放下来，而不改变所报告的位置（这样的一系列操作常被称为“抚摸”鼠标），鼠标的相对特征意味着计算机必须保存“当前鼠标位置”，鼠标移动的时候会改变它。

3. 跟踪球

跟踪球经常被描述成上下颠倒的机械鼠标。在套中自由旋转的跟踪球的运动由电位计或者轴编码器感应，用户通常用手掌挨着跟踪球旋转它。各种开关安装在跟踪球上手指够得到的地方，开关的用法和鼠标或者输入板手持光标定位器的按钮类似。



图4-15 带第三个自由度的游戏杆，这种游戏杆可以做顺时针及逆时针扭动。（由Measurement Systems, Inc.提供。）

4. 游戏杆

游戏杆（图4-15）可以左右移动或者上下移动，也是使用电位计来感应游戏杆的移动，经常用弹簧来使游戏杆恢复到原来的中心位置。一些游戏杆，包括图示的这种，有第三个自由

度：杆可以顺时针或逆时针扭动。

使用游戏杆直接控制屏幕光标的绝对位置是非常困难的，因为（通常）较短的轴上微小的移动会放大成光标5倍或者10倍的移动。这会使屏幕光标的移动急促跳动，很难快速精确地定位。因此游戏杆经常用来控制光标移动的速率而不是绝对光标位置。这意味着屏幕光标的当前
153~155 位置以游戏杆确定的速率改变。

5. 触摸板

鼠标、跟踪球和游戏杆都要占据工作台空间，而触摸板允许用户使用手指直接指向屏幕并在屏幕上移动光标。不同的触摸板使用了几种不同的技术。低分辨率的触摸板（每个方向上有10~50个可分解的位置）使用一系列的红外线发光二极管和光传感器（光敏二极管或者光敏晶体管）在显示区域形成看不见的光线网格，触摸屏幕会打断一条到两条光线束，因此通过被打断光线束就可以确定手指的位置。如果两条平行的光线束被打断，则认为手指处在这两条光线的中间，如果只有一条光线被打断，则认为手指处在这条光线上。

电容耦合的触摸板可以在每个方向上提供100个可分解的位置，当用户接触涂有导体膜的玻璃板时，电路从导体膜阻抗的改变来探测接触的位置[INTE85]。

触摸板最重要的参数是分辨率、激活所必需的压力（对光线触摸板这个不是问题）和透明度（这个对光线触摸板也不是问题）。对一些技术来说，一个重要的问题是视差：如果触摸板离显示器1/2英寸远，那么用户接触的位置是用户的眼睛和显示器上目标点对齐的位置，而不是接触板上直接垂直于目标点的位置。

用户习惯于得到可感知的反馈，但是触摸板并没有提供反馈。因此其他形式的立即反馈就很重要，如听得到的声音或者特定目标或位置的高光显示。

4.5.2 键盘设备

数字字母键盘是原始的文本输入设备，许多种不同的技术被用来探测键的按下，包括机械接触开关，容积变化以及磁场耦合等。键盘设备最重要的功能特征是它创造一个和按下的键惟一对应的码（如ASCII），有时候在数字字母键盘上需要允许和弦（一次按下几个键），让有经验的用户可以迅速地使用许多不同的命令。通常在标准**编码键盘**上这是不可能的，因为每次击键它返回的是一个ASCII码，如果两个键同时按下时它什么也不返回（除非附加的键是Shift，Ctrl或者其他特殊的键）。相反，**未编码键盘**返回同时按下的所有键的标志，因而允许和弦。

4.5.3 定值设备

大多数提供标量值的定值设备是基于电位计的，就像立体声音响中的音量和声调控制一样。定值设备通常是旋转电位计（刻度盘），电位计一般8个或10个一组安装在一起。简单的旋转电
156 位计可以旋转330°，这可能提供不了足够的范围和分辨率，连续旋转的电位计可以自由地朝两个方向旋转，因此在范围上没有限制，线性电位计显然是有限制的，它很少在图形系统中使用。

4.5.4 选择设备

功能键是最普通的选择设备，有时候它们制造成独立的单元，但更通常的情况是和键盘集成在一起。其他的选择设备是**按钮**，在许多输入板的手持光标上或者鼠标上都有。选择设备通常用来为图形程序输入命令或者菜单选项。专用的系统可以使用贴有永久标签的功能键，标签也可以是可替换的或者说“软”的。功能键可以在按钮旁边或者按键上包含一个小LCD或者LED显示。另外一种选择是把按钮安装在显示器的边角上，以便按钮标签在显示器上显示出来，正好挨着实际的按钮。

4.6 图像扫描仪

尽管数据输入板可以手工地数字化现有的线框图，但是这是一个缓慢而乏味的过程，不适用于复杂的图，而且它还不能数字化半色调的图像。图像扫描仪提供了有效的解决方法，一个电视摄像头和一个数字帧捕捉器的结合是不昂贵的方法，它可以获得中等分辨率（1000 × 1000，多种亮度级别）的黑白或彩色光栅图像。慢扫描电荷耦合器件（CCD）电视摄像头可以在30s内创建一幅2000 × 2000的图像。更便宜的方法是使用扫描头，扫描头由感光单元的网格组成，它安装在打印机的打印头上，它以每英寸80个单元的分辨率扫描图像。但是这些扫描分辨率是高品质的出版工作所不能接受的。出版中使用的是**照片扫描仪**（photo scanner），照片安装在一个旋转的鼓上，一束精确校准的光线被引导到照片上，反射光被光电管所测量。对于照相底片，透射光由鼓内的光电管测量，鼓是透明的。当鼓旋转时（图4-16），光源缓慢地从一端移动到另一端，从而对整个照片进行了光栅扫描。扫描彩色照片时，扫描过程要进行多趟，每次在光电管前面使用不同的滤镜把各种颜色分离出来。最高分辨率的扫描仪使用激光光源，其分辨率高于2000单元/英寸。

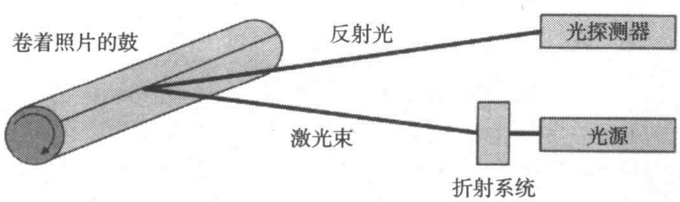


图4-16 照片扫描仪。光源沿鼓轴偏转，测量出偏转的光量

另一类扫描仪使用一细长条的CCD（叫作CCD阵列）。让图画从CCD阵列下面经过就可以将它数字化，根据所需要的分辨率来加快或减慢图画的移动。因此花1到2分钟走一趟，就可以数字化一幅大图。CCD阵列的分辨率为200~1000单元/英寸，低于照片扫描仪的分辨率。

线框图使用上面描述的任何一种方法都可以很容易地扫描，困难的地方是从扫描结果的一大堆像素中提取出有意义的信息来，**向量化**（vectorize）就是从光栅图像中提取直线、字符以及其他几何图元的过程。这个任务需要合适的算法而不是扫描硬件，向量化本质上是一个图像处理的问题，它分为以下几步：首先使用取阈值和边缘增强来整理光栅图像，消除图像中的污迹和污点，填补裂缝。然后使用特征提取算法把相邻的都为“打开”的像素合并在一起形成几何图元，如直线。在复杂的第二步，模式识别算法用来把简单的图元合并成圆弧、字母、符号等等，可能需要用户交互来消除由于中断的线条、暗污迹和许多邻近的直线交叉而引起的歧义性。

更困难的问题是将几何图元的集合组织成有意义的数据结构，当给CAD或者地形学（地图绘制）应用程序作输入时，无组织的线条集合没有太大用处。图画中表示高级的几何结构需要识别，因此，定义一块国土边界的轮廓线应该组织成多边形图元，代表圆弧的圆心的“+”要和圆弧本身组织在一起。这些问题已经有了部分解决。尽管算法在不断地改善，当情况困难时，商业系统还要依靠用户干预。

习题

4.1 如果长余辉荧光物质降低停闪频率，为什么不例行公事地使用它们？

4.2 编写一个在光栅显示器上显示测试图案的程序，必须提供三种不同的图案：(1) 宽度为1的

水平线，分别间隔0、1、2、3个像素；(2) 宽度为1的垂直线，分别间隔0、1、2、3个像素；(3) 一个像素的点的网格，网格间隔为5个像素。每个图案都要能用白色、红色、绿色、
157~158 蓝色以及交替颜色带来显示。显示图案时你所观察到的东西与光栅分辨率的讨论如何联系到一起？

4.3 假设每8个像素压缩到一个字节中，字节能以100 000字节/秒的速度传输并解压缩，装入一个$512 \times 512 \times 1$的位图需要多少时间？$1024 \times 1280 \times 1$的位图呢？

4.4 设计一个硬件单元的逻辑，完成二维光栅地址到字节地址加上字节中位地址的转换工作。给该单元的输入如下：(1)（x, y），光栅地址；(2) *base*，第0位包含光栅地址（0, 0）的内存字节的地址；(3) x_{max}，最大光栅x地址（最小为0）。单元的输出如下：(1) *byte*，其中某位中包含光栅地址（x, y）的字节的地址；(2) *bit*，在*byte*中包含（x, y）的位号。如果x_{max}
159 +1是2的幂时可以做什么简化？

第5章 几何变换

本章将介绍图形学中基本的二维和三维几何变换。在这里讨论的平移、缩放和旋转变换对于许多图形学的应用都非常基本，在后面的各章中还要经常出现。

变换被应用程序直接使用，或者用在图形子程序包中。一个城市规划应用程序使用平移变换将表示建筑或树木的符号放在合适的位置上，用旋转变换调整符号的朝向，用缩放变换改变符号的大小。总之，许多图形应用程序在绘图时使用几何变换改变物体（也称为**符号**或**模板**）的位置、朝向和尺寸。在第6章，三维旋转、平移和缩放变换将被用作生成三维物体的二维投影过程的一部分。在第7章中，我们将看到当今的图形软件包如何使用变换作为其实现的一部分并且使应用程序能够使用它们。

5.1 数学基础

本节回顾了本书所用的大部分数学知识，特别是向量和矩阵。这只是简要的介绍，而不是像线性代数和几何的教科书那样进行详细的讨论。但是，本节中我们假定读者已学过一年的大学数学课程，只是对几何和代数已有些淡忘了。如果对本节所包含的内容觉得十分容易，那么 161
可以直接跳到5.2节。如果这里叙述的概念不是近来的或者读者以前没有见到过，那么我们希望把本节作为您阅读本书其余部分的一个详细手册来参考。对于这里复习的概念，如果读者并不能确定能否掌握和应用，那么可确定的是：我们这里讨论的向量和矩阵运算与其等价的代数形式相比是一个简单的简化符号表示。由于这种表示十分方便和高效，因此我们竭力建议读者花些时间掌握它。我们已经找到一个很好的学习工具，如程序Mathematica™[WOLF91]——它允许你交互地执行向量和矩阵的符号或数值运算。如果读者有兴趣详细了解这些内容，请参阅[BANC83; HOFF61; MARS85]。

5.1.1 向量及其性质

你第一次接触到向量这个概念可能是在学习物理或者力学课程时，或许使用一个棒球离开球棒时的飞行速度和方向作为例子。在那一时刻，球的状态可以用一个带有箭头的直线段来表示。该直线段的箭头指示了球的运动方向，其长度表示球的速度。这个有向线段代表了球的*速度向量*。速度向量只是物理问题中出现的许多这类向量的一个例子。向量的其他例子如力、加速度和力矩。

已经表明向量的概念对于物理和数学是非常重要的。我们在计算机图形学中将广泛地使用向量。我们用它来表示世界坐标数据集中点的位置、空间中表面的方向、光线和实体、透明物体相互作用时的方向，及其他许多用途。我们将在后续的几章中遇到向量。当我们叙述它们的性质时，请注意我们将在何处及如何使用向量。

尽管一些教科书只从几何学的角度来叙述向量，我们将强调其双重性质，并使用它们的代数定义。这个定位引导我们采用简洁和紧凑的线性代数公式来表示，这是计算机图形学的数学问题所偏爱使用的。因此，我们将给出几何解释，作为理解某些概念的一种辅助。

有许多严格的定义，对于我们的目的而言，我们定义一个**向量**为实数的n元组，其中二维空间n是2、三维空间n是3、依此类推。我们将用粗体字母来表示向量[⊖]，本节中常用**u**、**v**或**w**表示。 162

⊖ 虽然不同文献中使用的符号并不一致，但在本书中，我们尽可能坚持采用研究文献中出现的惯用符号。由于在实践中的情况变化，读者将看到书中有时用大写字母、有时用小写字母来表示向量。

两种操作就是向量加法和实数与向量相乘的**标量乘法**[⊖]。这些操作具有特定的性质。加法操作必须满足交换律、结合律，而且有一个向量（通常称为**0**），它的特点是：对任何一个向量**v**，**0** + **v** = **v**。这些操作还具有可逆性（即，对任何一个向量**v**，必有一个向量**w**满足**v** + **w** = **0**；**w**被写为“ – **v**”）标量乘法必须满足下列规则：$(\alpha\beta)\mathbf{v} = \alpha(\beta\mathbf{v}), 1\mathbf{v} = \mathbf{v}, (\alpha + \beta)\mathbf{v} = \alpha\mathbf{v} + \beta\mathbf{v}, \alpha(\mathbf{v} + \mathbf{w}) = \alpha\mathbf{v} + \alpha\mathbf{w}$。

加法定义为各个相应的分量相加，而标量乘法也是对各个分量分别相乘。向量以垂直的方式书写，于是一个三维向量的例子如下：

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

而其向量加法，如下所示：

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}$$

计算机图形学大多数是在二维空间、三维空间或将在5.7节和6.6.4节中看到的四维空间中考虑的。

在向量和点之间做一个区分是重要的。当我们考虑一个点被定义为画到该点的向量时，这种表示要求我们指定该向量是从哪儿开始画起的，也就是要定义我们通常称为原点的地方。有许多操作能够在点上执行，但如不指定原点，则有一些向量操作在应用到该点时是没有定义的，如乘法操作。因此，最好不要混淆向量和点的性质，而把它们作为两个不同的概念来对待。无论如何，通常采用下列情况下的操作是更好的，即从两个点的差来形成向量。

有了上面关于向量和点间区分的讨论，我们可以探索将点作为向量对待的许多有用的性质。作为一个例子，考虑一个有指定原点的二维空间。我们可以根据熟知的**平行四边形法则**，向量加法定义为：向量**v**和**w**的相加，就是先从原点出发画一条带箭头的直线到**w**，然后，平移该直线使它的起始点落在点**v**上，则此时箭头端点就为**v** + **w**的向量。如果再画一条从原点到**v**的带箭头的直线，并且也对它进行类似的平移操作，我们就得到了一个平行四边形，如图5-1所示。一个实数α与一个向量相乘的标量乘法的定义也是类似的：从原点到**v**画一条带箭头的直线，根据α的值改变这条直线的长度，而该直线的起始点保持不变，则直线的箭头端点即为所定义的$\alpha\mathbf{v}$。显然，在实数域的三维欧氏空间中，向量加法和纯量乘法的定义也是相同的。

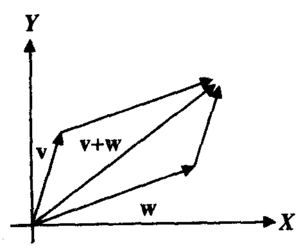


图5-1 平面上的向量加法运算

假设一个向量空间中有上述两种操作，那么用向量就可以很
163 方便地处理一些事情。其中之一是线性组合。向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$的线性组合就是可以表示成后面形式的任何向量：$\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_n\mathbf{v}_n$。向量的线性组合可以用来描述许多对象。第9章中我们将遇到线性组合在曲线和曲面中的应用。

5.1.2 向量点积

给出两个n维向量

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \quad 和 \quad \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix}$$

⊖ 标量（或实数）将用希腊字母表示，特别是用那些在字母表中靠前的一些字母。

我们定义它们的**点积**或**内积**为 $x_1y_1+\cdots+x_ny_n$。向量**v**和**w**的点积通常表示为**v** · **w**。

平面上的点(x, y)到原点（0, 0）的距离是$\sqrt{x^2+y^2}$。一般地，n维空间中从点（$x_1,\cdots,x_n$）到原点的距离是$\sqrt{x_1^2+\cdots+x_n^2}$。如果我们设向量**v**为

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

则它到原点的距离就是$\sqrt{\mathbf{v}\cdot\mathbf{v}}$。这就是我们对$n$维向量**长度**的定义。我们将此长度表示为$\|\mathbf{v}\|$。在标准的$n$维空间中，两个点之间的距离的定义是类似的：$P$和$Q$之间的距离就是向量$Q-P$的长度。

5.1.3 点积的性质

点积有几个很好的性质。第一，它是对称的：$\mathbf{v}\cdot\mathbf{w}=\mathbf{w}\cdot\mathbf{v}$。第二，它**不会退化**：只有当$\mathbf{v}=\mathbf{0}$时，$\mathbf{v}\cdot\mathbf{v}=0$。第三，它满足**双线性**关系：$\mathbf{v}\cdot(\mathbf{u}+\alpha\mathbf{w})=\mathbf{v}\cdot\mathbf{u}+\alpha(\mathbf{v}\cdot\mathbf{w})$。

点积可以用来生成长度为1的向量（这称为对**向量进行规格化**）。对向量**v**进行规格化，只需要简单地计算$\mathbf{v}'=\mathbf{v}/\|\mathbf{v}\|$。这样，所得的向量的长度是1，并被称为**单位向量**。

点积也能用来度量角度。**向量v和w之间的夹角是**

$$\cos^{-1}\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\ \|\mathbf{w}\|}\right)$$

注意，如果**v**和**w**是单位向量，则上式中的除法就不必要了。

如果我们有一个单位向量**v'**和另一个向量**w**，并将**w**垂直地投影到**v'**上，如图5-2所示，所得结果是**u**，那么**u**的长度应当是**w**的长度与$\cos\theta$ 相乘的结果，此处的θ是**v**和**w**之间的夹角。这也就是 164
说：

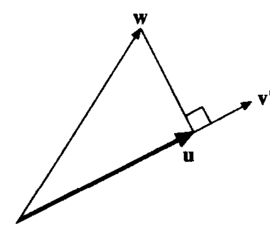


图5-2 **w**投影到单位向量**v'**上的结果是向量**u**，该向量的长度是$\|\mathbf{w}\|$乘以**v'**和**w**之间夹角的余弦

$$\begin{aligned}\|\mathbf{u}\| &= \|\mathbf{w}\|\cos\theta \\ &= \|\mathbf{w}\|\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\ \|\mathbf{w}\|}\right) \\ &= \mathbf{v}\cdot\mathbf{w}\end{aligned}$$

因为**v**的长度是1。这样，就对点积给出了一个新的诠释：假设**v**是单位向量，则**v**和**w**的点积就是**w**在**v**上投影的长度。我们会遇到点积的许多应用，特别是在第14章中。在第14章中点积被用于描述光与平面如何相交。

5.1.4 矩阵

矩阵就是将数排列成矩形样子的一个数组，我们在点和向量运算中经常会用到它。可以把矩阵看成是表示按线性变换对其操作数进行变换的规则。它的每一个元素（通常是实数）都有两个序号，按照惯例，第一个序号是关于行的，而第二个序号是关于列的。按照数学上的习惯，这些序号都从1开始记数；但有一些程序语言是将序号从0开始记数。对于用那些程序语言的程序员而言，他们要将所有序号移动1位。因此，如果**A**是一个矩阵，那么$a_{3,2}$是指第3行第2列的元素。当运用符号形式的序号时，比如说a_{ij}，则序号之间的逗号被省略。

n维空间中我们写成如下形式的向量：

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

可以认为它是一个$n\times 1$的矩阵，并称为**列向量**。我们在本书的很多地方使用矩阵及其相关的操作（5.1.5节至5.1.8节）。矩阵在几何变换（本章）、三维观察（第6章）、三维图形软件包（第7章）和曲线及曲面描述（第9章）中起着很重要的作用。

5.1.5 矩阵乘法

矩阵的乘法是按照下列的原则进行的：如果**A**是一个$n\times m$的矩阵，其元素为a_{ij}，而**B**是一个$m\times p$的矩阵，元素是b_{ij}，那么**AB**就是一个$n\times p$的矩阵，元素是c_{ij}，且$c_{ij}=\sum_{s=1}^{m}a_{is}b_{sj}$。如果将**B**的列都当成单独的向量，$\mathbf{B}_1, \cdots, \mathbf{B}_p$，而**A**的行也当成单独的向量$\mathbf{A}_1, \cdots, \mathbf{A}_m$（旋转90°后就恢复成水平的了），那么，我们看到$c_{ij}$就是$\mathbf{A}_i\cdot\mathbf{B}_j$。除了交换律外，矩阵乘法保持了一般乘法的其他特
165 点。一般而言，**AB**和**BA**是不同的。当然，乘法满足相对于加法的分配律：$\mathbf{A}(\mathbf{B}+\mathbf{C})=\mathbf{AB}+\mathbf{AC}$。对于乘法，有一个恒定的元素，称为**单位矩阵I**，它是一个正方形的矩阵，除了对角线上的元素为1之外，其余的元素均为0（例如，对于元素δ_{ij}，除$i=j$时$\delta_{ii}=1$之外，其他项$\delta_{ij}=0$）。矩阵乘法的图形描述见例5.1。

5.1.6 行列式

一个方形矩阵的**行列式**就是一个数，它是从矩阵的元素形成的。行列式的计算有点复杂，因为它是递归定义的。2×2的矩阵$\begin{bmatrix} a & c \\ b & d\end{bmatrix}$的行列式就是$ad-bc$。一个$n\times n$的矩阵的行列式是在比它小的矩阵的行列式的基础上进行计算的。假设A_{1i}是将$n\times n$的矩阵A的第一行和第i列的元素去掉以后所得的$(n-1)\times(n-1)$矩阵的行列式，那么，行列式A是由下面的式子定义的：

$$\det \mathbf{A}=\sum_{i=1}^{n}(-1)^{1+i}A_{1i}$$

在三维空间中，行列式有一种特殊的应用：*叉积*。两个向量

$$\mathbf{v}=\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad 和 \quad \mathbf{w}=\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

的叉积是运用矩阵的行列式进行计算的，即

$$\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

其中，字母**i**, **j**, **k**表示沿三个坐标轴方向的单位向量。计算的结果是关于变量**i**, **j**, **k**的一个线性组合；此时，这些变量可以分别替换成向量$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$。于是，所得的结果为向量

$$\begin{bmatrix} v_2w_3-v_3w_2 \\ v_3w_1-v_1w_3 \\ v_1w_2-v_2w_1 \end{bmatrix}$$

表示成$\mathbf{v}\times\mathbf{w}$。该向量的一个特点是垂直于由$\mathbf{v}$和$\mathbf{w}$所定义的平面，而它的长度是$\|\mathbf{v}\|\,\|\mathbf{w}\|\,|\sin\theta|$，其中，$\theta$是$\mathbf{v}$和$\mathbf{w}$之间的夹角。在第9章中我们将探讨叉积的性质。第9章中我们用叉积来确定多边形的平面方程。

5.1.7 矩阵的转置

一个$n\times k$的矩阵相对于它的对角线（从左上到右下）进行对称的换位，就得到了一个$k\times n$的矩阵。第一个矩阵中的元素排列是a_{ij} $(i=1,\cdots,n;\ j=1,\cdots,k)$，所得的矩阵中的元素排列是$b_{ij}$ $(i=1,\cdots,k;\ j=1,\cdots,n)$，且$b_{ij}=a_{ji}$。这个新的矩阵称为原来矩阵的**转置**。矩阵$\mathbf{A}$的转置写为$\mathbf{A}^{\mathrm{T}}$。如果我们将$n$维空间中的一个向量当成是一个$n\times 1$的矩阵，那么它的转置就是一个$1\times n$的矩阵 166
（有时称为行向量）。运用转置，我们可以对点积给出一种新的表达，即$\mathbf{u}\cdot\mathbf{v}=\mathbf{u}^{\mathrm{T}}\mathbf{v}$。

5.1.8 矩阵的逆

矩阵乘法与一般的乘法不同：一个矩阵可能没有相应的逆。事实上，逆只是对方形矩阵定义的，并且也不是所有的方形矩阵都有逆。只有那些其行列式不为零的方形矩阵才有逆。

如果$\mathbf{A}$和$\mathbf{B}$都是$n\times n$的矩阵，并且$\mathbf{AB}=\mathbf{BA}=\mathbf{I}$，其中$\mathbf{I}$是$n\times n$的单位矩阵，那么，$\mathbf{B}$称为是$\mathbf{A}$的逆，并写为$\mathbf{A}^{-1}$。对于元素是实数的$n\times n$的矩阵，只要$\mathbf{AB}=\mathbf{I}$或者$\mathbf{BA}=\mathbf{I}$成立（只要有一个成立，那么另一个也成立），就足以证明它们是可逆的。

如果我们有一个$n\times n$的矩阵，求它的逆的较好的方法是高斯消元法，特别是对于大于3×3的矩阵。关于这种方法（包括实现的有效程序）的好的参考文献是[PRES88]。

例5.1 我们将在5.7节看到4×4矩阵在计算机图形学中的重要性；它们被广泛地用于3D变换。

问题：

a. 计算矩阵A和矩阵B的积$C=AB$，其中

$$A=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&\frac{1}{k}&1\end{bmatrix}\quad 和 \quad B=\begin{bmatrix}\cos\theta&0&\sin\theta&0\\0&1&0&m\\-\sin\theta&0&\cos\theta&n\\0&0&0&1\end{bmatrix}$$

我们将在6.5节及5.7节发现这些矩阵分别指定了透视投影、旋转及两个平移。

b. 写一个C函数，它返回两个4×4 矩阵A和B的乘积C。

解答：

a. 按5.1.5节定义的矩阵乘法的规则，可说明如下：

$$c_{ij}=\sum_{s=1}^{m}a_{is}b_{sj}\qquad\begin{bmatrix}b_{11}&b_{12}&b_{13}&b_{14}\\b_{21}&b_{22}&b_{23}&b_{24}\\b_{31}&b_{32}&b_{33}&b_{34}\\b_{41}&b_{42}&b_{43}&b_{44}\end{bmatrix}$$

$$\begin{bmatrix}a_{11}&a_{12}&a_{13}&a_{14}\\a_{21}&a_{22}&a_{23}&a_{24}\\a_{31}&a_{32}&a_{33}&a_{34}\\a_{41}&a_{42}&a_{43}&a_{44}\end{bmatrix}\quad\begin{bmatrix}c_{11}&c_{12}&c_{13}&c_{14}\\c_{21}&c_{22}&c_{23}&c_{24}\\c_{31}&c_{32}&c_{33}&c_{34}\\c_{41}&c_{42}&c_{43}&c_{44}\end{bmatrix}$$ 167

这里我们选择元素c_{43}予以求值。乘法公式表明，我们应该将矩阵A的第四行元素与矩阵B的第三列元素分别相乘。执行这个运算导致结果$c_{43}=\cos\theta/k$。对于C的每一个元素，应用这一过程，我们可以得到结果矩阵为：

$$C = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & m \\ 0 & 0 & 0 & 0 \\ \frac{-\sin\theta}{k} & 0 & \frac{\cos\theta}{k} & \frac{n}{k}+1 \end{bmatrix}$$

b.
```
typedef struct Matrix4Struct {
    double element[4][4];
}Matrix4;

/* 做矩阵乘法c = ab */
/* 注意c必须不指向两个输入矩阵中的任何一个 */
Matrix4 *V3MatMul(a, b, c)
Matrix4 *a, *b, *c;
{
    int   i, j, k;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            c->element[i][j] = 0.0;
            for (k = 0; k < 4; k++)
                c->element[i][j] +=
                    a->element[i][k] * b->element[k][j];
        }
    }
    return (c);
}
```

5.2 二维变换

我们通过给点的坐标增加平移量来实现将(x, y)平面上的点**平移**到新的位置。对于每一个点$P(x, y)$，在与x轴平行的方向上平移d_x且在与y轴平行的方向上平移d_y得到新点$P'(x', y')$，可以表示为

$$x' = x + d_x, \qquad y' = y + d_y \tag{5-1}$$

如果我们定义列向量

168

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{5-2}$$

那么式(5-1)可以被表示成更加简洁的形式

$$P' = P + T \tag{5-3}$$

对物体上的每个点应用式(5-2)，我们可以平移该物体。因为直线段由无限个点组成，所以这一过程将持续无限长的时间。幸运的是，我们只需要平移直线段的端点，然后在平移后的端点之间绘制新的直线段就可以平移直线段上的全部点；这对于缩放（拉伸）和旋转变换也是正确的。图5-3显示平移房子的轮廓线的效果，平移量为$(3, -4)$。

通过以下乘法运算，点可以实现在x轴方向**缩放**（拉伸）s_x且在y轴方向缩放s_y到新的位置：

$$x' = s_x \cdot x, \qquad y' = s_y \cdot y \tag{5-4}$$

用矩阵的形式表示为

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \text{ or } P' = S \cdot P \tag{5-5}$$

其中S是式(5-5)中的矩阵。

在图5-4中，房子在x轴方向被缩小为原来的1/2，在y轴方向缩小为原来的1/4。注意，缩放变换是相对于原点的：房子变小，更靠近原点。如果缩放因子比1大，房子会变大而且远离原点。相对于某一点而非原点的缩放变换将在5.3节中加以讨论。房子的比例已被改变：因为使用**不同的**缩放因子，$s_x \neq s_y$。使用**均匀**缩放，即$s_x = s_y$，房子的比例不受影响。

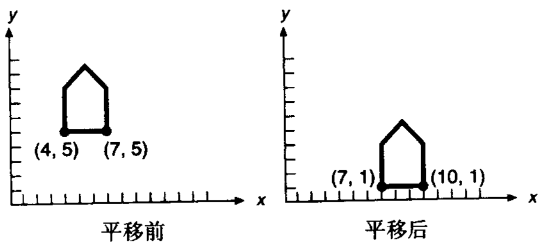


图5-3 房子的平移

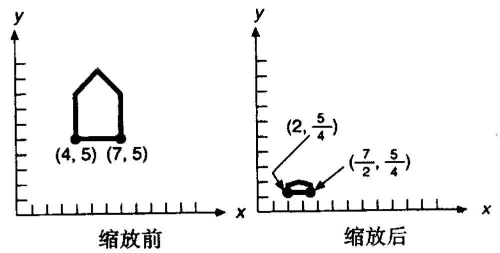


图5-4 房子的缩放，缩放是非均匀的，并且房子改变了位置

点可以绕原点**旋转**θ角。旋转在数学上被定义为

$$x' = x \cdot \cos\theta - y \cdot \sin\theta, \quad y' = x \cdot \sin\theta + y \cdot \cos\theta \tag{5-6}$$ 169

用矩阵的形式表示为

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{或} \quad P' = R \cdot P \tag{5-7}$$

其中R为式(5-7)中的旋转矩阵。图5-5显示旋转45°的房子。与缩放变换相似，旋转是绕原点进行的：绕任意点的旋转在5.3节中介绍。

正的旋转角度是从x轴向y轴**逆时针方向**测量。对于负的旋转角度（顺时针方向），恒等式$\cos(-\theta) = \cos\theta$和$\sin(-\theta) = -\sin\theta$用于修改式(5-6)和式(5-7)。

式(5-6)很容易从图5-6中提取出，在图5-6中一个角度为θ的旋转将$P(x, y)$变换为$P'(x', y')$。因为旋转绕原点进行，所以从原点到P和P'点的距离相等，在图中标记为r。通过简单的三角学知识，我们发现

$$x = r \cdot \cos\phi, \quad y = r \cdot \sin\phi \tag{5-8}$$

且

$$\begin{aligned} x' &= r \cdot \cos(\theta + \phi) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta \\ y' &= r \cdot \sin(\theta + \phi) = r \cdot \cos\phi \cdot \sin\theta + r \cdot \sin\phi \cdot \cos\theta \end{aligned} \tag{5-9}$$

将式(5-8)代入式(5-9)得到式(5-6)。

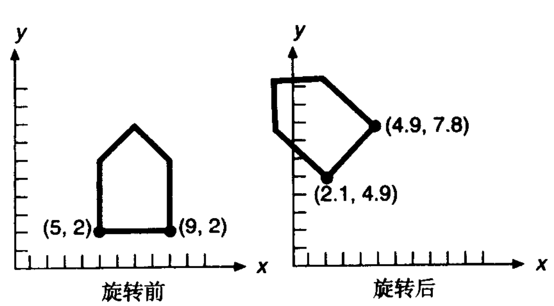


图5-5 房子的旋转，而且房子改变了位置

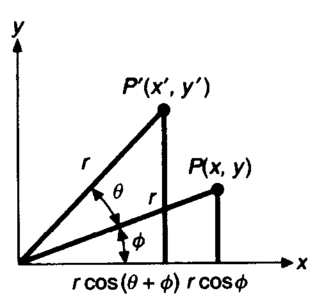


图5-6 旋转方程的推导

5.3 齐次坐标和二维变换的矩阵表示

平移、缩放和旋转的矩阵表示分别为

$$P' = T + P \tag{5-3}$$

170

$$P' = S \cdot P \tag{5-5}$$

$$P' = R \cdot P \tag{5-7}$$

可是，平移的处理方法（加法）不同于缩放和旋转（乘法）。我们更愿意用一种一致的方式处理三种变换，这样它们可以很容易地结合起来。

如果点被表示为**齐次坐标**（homogeneous coordinates），则这三种变换都可以用乘法处理。齐次坐标首先从几何中发展起来[MAXW46；MAXW51]，随后被应用于图形学中[ROBE65；BLIN77b；BLIN78a]。大量的图形子程序包和显示处理器使用齐次坐标和齐次变换。

在齐次坐标中，我们为每个点增加第三维坐标。每个点被表示为一个三元组(x, y, W)，而不是一对数(x, y)。与此同时，我们说两个齐次坐标(x, y, W)和(x', y', W')表示同一个点，当且仅当其中的一个是另一个的倍数。因此，(2, 3, 6)和(4, 6, 12)是用不同的三元组表示的同一个点。也就是说，每个点有许多不同的齐次坐标表示。同时，至少有一个齐次坐标值为非零：即(0,0,0)是不允许的。如果坐标W非零，我们可以用它去除齐次坐标：(x, y, W)与$(x/W, y/W, 1)$表示同一个点。当W非零时，我们通常做这个除法，数x/W和y/W被称为齐次点的笛卡儿坐标。$W = 0$的点被称为无穷远点，在我们的讨论中不经常出现。

坐标三元组通常表示三维空间中的点，但是在这里我们使用它们表示二维空间中的点。两者之间的联系是：如果我们取所有表示同一个点的三元组（也就是说，所有形式为(tx, ty, tW)的三元组，其中$t \neq 0$），我们得到三维空间中的一条直线。因此，每一个齐次点表示三维空间中的一条直线。如果我们**齐次化**（homogenize）点（除以W），我们得到一个形式如$(x, y, 1)$的点。所以，齐次化的点形成一个被等式$W = 1$定义的(x, y, W)空间中的平面。图5-7显示了这种关系。无穷远点不在该平面上。

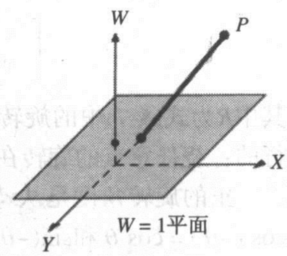


图5-7 *XYW*齐次坐标空间，带有$W = 1$平面和投影到$W = 1$平面上的点$P\ (X, Y, W)$

因为现在点表示为三个元素的列向量，所以用于乘以一个点向量来产生另一个点向量的变换矩阵必须是3 × 3的矩阵。在齐次坐标的3 × 3矩阵形式中平移公式(5-1)为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-10}$$

我们提醒读者注意，一些图形学的教科书，包括[FOLE82]，使用左乘行向量矩阵的习惯，而不是右乘列向量。从一种习惯到另一种习惯变换矩阵必须被转置，正如行向量和列向量互为转置：

$$(P \cdot M)^{\mathrm{T}} = M^{\mathrm{T}} \cdot P^{\mathrm{T}}$$

171 式(5-10)可以被表示为另外一种形式

$$P' = T(d_x, d_y) \cdot P \tag{5-11}$$

其中

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-12}$$

如果一个点被平移$T(d_{x1}, d_{y1})$到达P'点，然后又被平移$T(d_{x2}, d_{y2})$到P''会发生什么呢？凭直觉我们期望结果是一个净平移$T(d_{x1}+d_{x2}, d_{y1}+d_{y2})$。为了确认这种直觉，我们从已知入手：

$$P' = T(d_{x1}, d_{y1}) \cdot P \tag{5-13}$$

$$P'' = T(d_{x2}, d_{y2}) \cdot P' \tag{5-14}$$

现在，将式(5-13)代入式(5-14)，我们得到

$$P'' = T(d_{x2}, d_{y2}) \cdot (T(d_{x1}, d_{y1}) \cdot P) = (T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})) \cdot P \tag{5-15}$$

矩阵乘积$T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})$为

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-16}$$

净平移确实是$T(d_{x1}+d_{x2}, d_{y1}+d_{y2})$。矩阵乘积被不同地称为$T(d_{x1}, d_{y1})$和$T(d_{x2}, d_{y2})$的**复合**（compounding）、**连接**（catenation）、**串联**（concatenation）或**合成**（composition）。这里，我们将规范地使用术语合成。

类似地，缩放式(5-4)以矩阵形式表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-17}$$

定义

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-18}$$

我们有

$$P' = S(s_x, s_y) \cdot P \tag{5-19}$$

如同净平移是加法，我们希望连续缩放是乘法。已知

$$P' = S(s_{x1}, s_{y1}) \cdot P \tag{5-20}$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P' \tag{5-21}$$

那么，将式(5-20)代入式(5-21)，我们得到 172

$$P'' = S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1}) \cdot P) = (S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})) \cdot P \tag{5-22}$$

矩阵乘积$S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})$为

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-23}$$

因此，连续的缩放变换也确实是乘法。

最后，旋转式(5-6)可以表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-24}$$

令

$$R(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{5-25}$$

我们有

$$P'=R(\theta)\cdot P \tag{5-26}$$

两个连续旋转是加法的证明留为习题5.2。

在式(5-25)的左上角的2×2子矩阵中，将矩阵的两行的每一行都看成向量。这两个向量有如下三个特性：

1) 每个都是单位向量。

2) 两个向量正交（它们的点积为0）。

3) 第一个向量和第二个向量分别旋转$R(\theta)$，将得到x轴和y轴的正方向（因为条件1和条件2的存在，这一性质等价于子矩阵的行列式为1）。

前两个特性对2×2子矩阵的列向量也成立。两个方向则是向量沿着x轴和y轴正方向旋转到的方向。这些特性表明当我们知道想要经过旋转得到的效果时，有两种有用的提取旋转矩阵的方法。一个具有这些特性的矩阵被称为**特殊正交矩阵**。

一个变换矩阵的形式如下：

$$\begin{bmatrix}r_{11} & r_{12} & t_x\\ r_{21} & r_{22} & t_y\\ 0 & 0 & 1\end{bmatrix} \tag{5-27}$$

其中左上角2×2子矩阵是正交的，该变换矩阵保持角度和长度不变。也就是说，一个单位正方
173 形仍然保持为单位正方形，既不会变成具有单位边长的菱形，也不会变成非单位边长的长方形。这样的变换也被称为**刚体变换**（rigid-body transformation），因为变换后的物体在任何情况下都不会扭曲。任意的旋转和平移矩阵序列都将产生这种形式的矩阵。

任意的旋转、平移和缩放矩阵序列的乘积如何呢？它们被称为**仿射变换**（affine transformation），并且具有保持直线平行性的特性，但是不保持长度和角度不变。图5-8显示对一个单位正方形施加−45°角的旋转和非均匀缩放后的结果。很明显角度或长度在序列中都发生变化，但是平行线仍然平行。进一步的旋转、缩放和平移操作都不会导致平行线不平行。$R(\theta)$、$S(s_x, s_y)$和$T(d_x, d_y)$也是仿射变换。

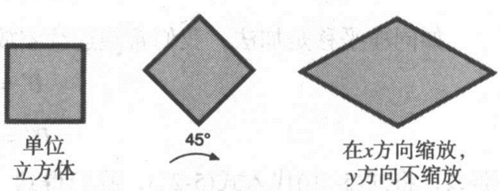


图5-8 单位立方体被旋转−45°，并且被非均匀缩放。结果是单位立方体的仿射空间，在这个空间中保持了线的平行性，但角度和长度都没有保持

另一种类型的基本变换**错切变换**（shear transformation）也是仿射变换。二维错切变换分为两类：沿x轴错切和沿y轴错切。图5-9显示沿每个轴错切正方形的效果。该操作用矩阵表示为

$$SH_x=\begin{bmatrix}1 & a & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{5-28}$$

错切矩阵中项a是比例常数。注意，乘积$SH_x\,[x\ y\ 1]^T$等于$[x+ay\ y\ 1]^T$，清楚地显示在x方向上的比例变化是y的函数。

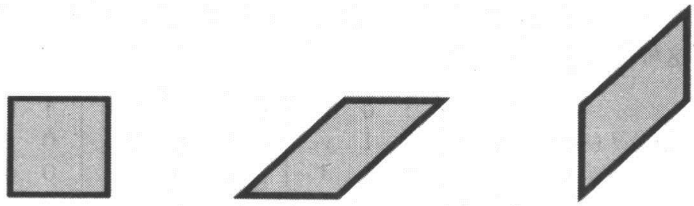


图5-9 应用于单位立方体的图元错切操作。在每种情况中，斜线的长度现在都大于1

类似地，矩阵

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-29}$$

沿y轴错切。

5.4 二维变换的合成

合成的观点在前面的部分已经介绍过。这里，我们使用合成将基本的R、S和T矩阵结合起来，以产生所需的一般的结果。合成变换的基本目的是通过对一个点施加单个合成后的变换来获得效率，而不是一个接一个地应用一系列变换。

考虑物体绕某个任意点P_1旋转。因为我们仅知道如何绕原点旋转，我们将原来（困难的）的问题转换为三个分开（容易的）的问题。因此，为了绕P_1旋转，我们需要三个基本变换的序列：

1) 平移使得P_1位于原点。

2) 旋转。

3) 平移使得P_1回到原来的位置。

这个序列如图5-10所示，在图5-10中房子绕$P_1(x_1, y_1)$旋转。首先平移$(-x_1, -y_1)$，反之后面反向平移(x_1, y_1)。最终结果与仅施加旋转操作不同。

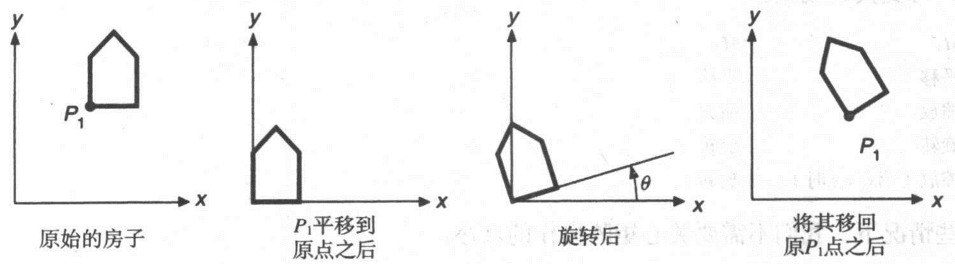


图5-10 房子绕着点P_1以角度θ旋转 174~175

净变换（net transformation）为

$$T(x_1, y_1)\cdot R(\theta)\cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta)+y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta)-x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5-30}$$

类似的方法用于相对任意点P_1缩放物体。首先，平移使得P_1到达原点，然后进行缩放，最后平移回到P_1。在这种情况下，净变换为

$$T(x_1,y_1)\cdot S(s_x,s_y)\cdot T(-x_1,-y_1)=\begin{bmatrix}1&0&x_1\\0&1&y_1\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&-x_1\\0&1&-y_1\\0&0&1\end{bmatrix}$$

$$=\begin{bmatrix}s_x&0&x_1(1-s_x)\\0&s_y&y_1(1-s_y)\\0&0&1\end{bmatrix} \tag{5-31}$$

假设我们希望缩放、旋转和设置房子的位置如图5-11所示，并以P_1为旋转和缩放的中心。变换序列为将P_1平移到原点，进行缩放和旋转，然后从原点平移到放置房子的新位置P_2。记录这个变换的数据结构可能包括缩放因子、旋转角度和平移量，以及变换的次序，或者简单地记录合成变换矩阵：

$$T(x_2,y_2)\cdot R(\theta)\cdot S(s_x,s_y)\cdot T(-x_1,-y_1) \tag{5-32}$$

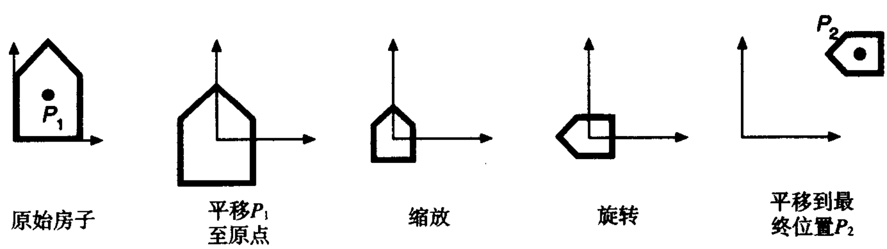


图5-11　房子绕P_1旋转，并将在P_1放置的房子放在P_2位置

176 如果M_1和M_2表示基本的平移、缩放或旋转，什么时候$M_1\cdot M_2=M_2\cdot M_1$？也就是说，什么时候$M_1$和$M_2$可交换？通常，矩阵乘法当然是不可交换的。但是，很容易证明，在下列特殊情况下，可交换性成立：

M_1	M_2
平移	平移
缩放	缩放
旋转	旋转
缩放（当$s_x=s_y$时）	旋转

在这些情况下，我们不需要关心矩阵操作的次序。

5.5　窗口到视口的变换

一些图形软件包允许程序员在浮点**世界坐标**系下指定输出图元的坐标，使用对应用程序有意义的单位：埃，微米，米，英里，光年等等。使用术语“世界”是因为应用程序正在表示一个正在被交互创建或向用户显示的世界。

假定输出图元在世界坐标系下指定，则必须告诉图形子程序包如何将世界坐标系映射到屏幕坐标系（我们使用特殊的术语*屏幕坐标*以便将这部分与SRGP联系起来，但是因为可能使用硬拷贝输出设备，在这种情况下术语*设备坐标*更加准确）。我们可以通过由程序员向图形软件

包提供变换矩阵来实现这种映射。另一种方式是让程序员在世界坐标中指定一个矩形区域，称为**世界坐标窗口**，另外在屏幕坐标上指定一个对应的矩形区域，称为**视口**，世界坐标系窗口将被映射到视口中。将窗口映射到视口的变换被应用到世界坐标系中的所有输出图元，从而将它们映射到屏幕坐标系中。图5-12显示了这一概念。如图5-12所示，如果窗口和视口高度宽度比不同，就会出现一个非均匀缩放变换。如果应用程序改变窗口或视口，那么新绘制到屏幕上的输出图元会受到变化的影响，而现存的输出图元则不受这种变化的影响。

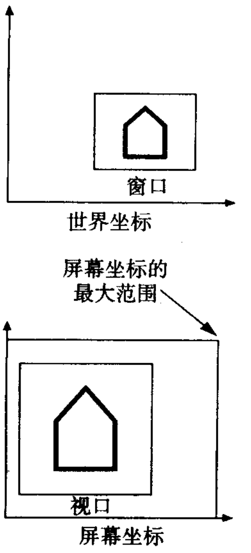


图5-12 世界坐标中的窗口和屏幕坐标中的视口确定了一个映射，该映射作用于世界坐标中的所有图元

177

修饰语*世界坐标*与*窗口*一起使用，以强调我们不是在讨论一个*窗口管理器窗口*，后者是一个不同的更新的概念，却不幸具有相同的名字。当不会产生歧义时，我们就丢弃这个修饰语。

如果SRGP提供世界坐标下的输出图元，视口将出现在当前的画布上，默认为画布0，即屏幕。应用程序能够在任何时刻改变窗口或视口，在这种情况下随后指定的输出图元将被施加新的变换。如果这种变化包括一个不同的视口，那么新的输出图元将被定位在画布上不同于原来输出图元的位置上，如图5-13所示。

一个窗口管理器可能将SRGP的画布0映射为全屏窗口的一部分，在这种情况下并不是整个画布或者视口都必须可见。

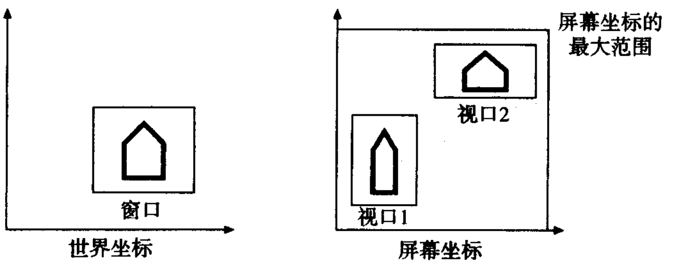


图5-13 以两个视口画输出图元的效果。描述房子的输出图元先被画在视口1，该视口变化成视口2，然后应用程序再次调用软件包来画输出图元

给定窗口和视口，什么是将窗口从世界坐标系映射到屏幕坐标系下的视口的变换矩阵？这个矩阵可以由一个三步的变换合成求出，如图5-14所示。第一步，用左上角点和右下角点表示的窗口被平移到世界坐标系的原点。第二步，窗口的尺寸被缩放成与视口尺寸相等。最后，用一个平移变换设置视口的位置。完整的矩阵M_{wv}（通过两个平移矩阵和一个缩放矩阵的合成获得）为：

$$M_{wv} = T(u_{min}, v_{min}) \cdot S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right) \cdot T(-x_{min}, -y_{min})$$

$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x_{min}\cdot\frac{u_{max}-u_{min}}{x_{max}-x_{min}}+u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y_{min}\cdot\frac{v_{max}-v_{min}}{y_{max}-y_{min}}+v_{min} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-33}$$

乘法$P = M_{wv}[x\ y\ 1]^T$给出期望的结果：

$$P = \left[(x-x_{min})\cdot\frac{u_{max}-u_{min}}{x_{max}-x_{min}}+u_{min} \quad (y-y_{min})\cdot\frac{v_{max}-v_{min}}{y_{max}-y_{min}}+v_{min} \quad 1\right] \tag{5-34}$$

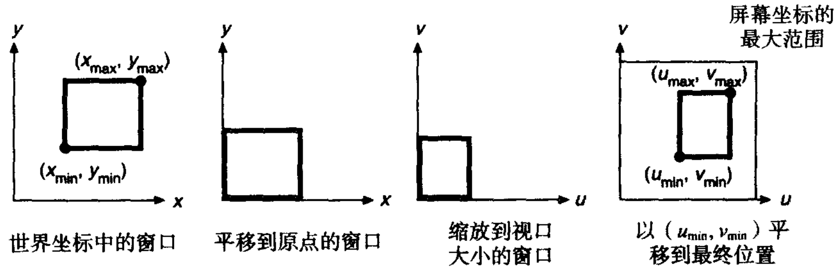


图5-14 变换世界坐标窗口到视口的步骤

许多图形软件包将窗口–视口变换与用窗口对输出图元的裁剪结合起来。裁剪的概念在第3章已经介绍过；图5-15显示了在窗口和视口环境下的剪裁。

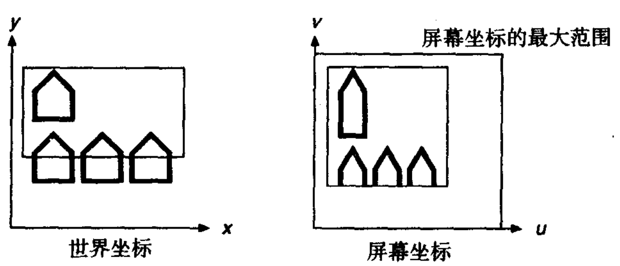


图5-15 世界坐标中的输出图元被窗口裁剪，保留的部分被显示在视口中

5.6 效率

R、S和T操作最普通的合成产生形式如下的矩阵：

178~179

$$M = \begin{bmatrix} r_{11} & r_{12} & t_z \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-35}$$

左上角2×2子矩阵是一个合成的旋转和缩放矩阵，而t_x和t_y是合成的平移。用3×3矩阵乘以向量来计算$M \cdot P$需要九个乘法和六个加法。然而，式(5-35)最后一行的固定结构将实际运算简化为

$$\begin{aligned} x' &= x\cdot r_{11} + y\cdot r_{12} + t_x \\ y' &= x\cdot r_{21} + y\cdot r_{22} + t_y \end{aligned} \tag{5-36}$$

将这一过程简化为四个乘法和四个加法——意义重大的加速，特别是当这个运算被应用到每张图上数以百计甚至千计的点时。所以，尽管3×3矩阵对合成二维变换很方便也很有用，但是通过利用最终矩阵的特殊结构我们能够在程序中最有效地利用它。一些硬件矩阵乘法器具有并行的加法器和乘法器，从而减少或消除了这个问题。

5.7 三维变换的矩阵表示

正如使用齐次坐标时二维变换可以表示为3×3的矩阵一样，如果我们也使用齐次坐标表示三维空间上的一个点，三维变换能够表示成4×4的矩阵。因此，我们用（x, y, z, W）表示一个点而不是用（x, y, z），其中如果一个四元组是另一个四元组的非零倍数，则两个四元组表示同一个点；四元组（0,0,0,0）是不允许的。与二维相同，点（x, y, z, W）的标准表示为（$x/W, y/W, z/W$, 1），其中$W\neq 0$。将点变换为这种形式被称为**齐次化**，如前所述。同样，W坐标为零的点被称为无穷远点。齐次化也有一个几何解释。每个三维空间上的点被表示成通过四维空间原点的一条直线，这些点的齐次化表示形成一个四维空间的三维子空间，该子空间可以由单个方程$W=1$定义。

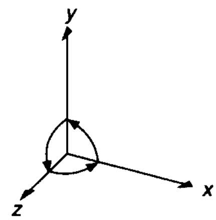


图5-16 右手坐标系

在本书中三维坐标系采用**右手系**，如图5-16所示。按照惯例，一个右手系的正向旋转是，当从一个坐标轴的正方向向原点看时，一个90°的*逆时针*旋转将一个轴的正方向变换为另一个轴的正向。下面的表从这个惯例得出：

旋转轴是	正的旋转方向是
x	y到z
y	z到x
z	x到y

图5-16描绘了这些正方向。读者需要注意，不是所有的图形学教科书都遵守这个惯例。

这里我们使用右手系统，因为它是标准的数学惯例，即使在图形学中考虑到添加在显示器表面的左手系统更加方便（见图5-17），180
因为一个左手系统给出一种自然的解释，即大的z值更加远离视点。注意到在左手系中，从坐标轴的正方向向原点看时正向旋转是*顺时针*。这种正向旋转的定义使得不论是右手还是左手系都可以使用本节给出的相同的旋转矩阵。从右手系统到左手系和相反的转换在5.9节中讨论。

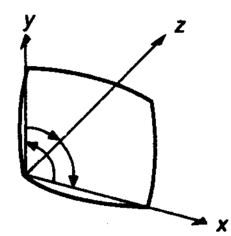


图5-17 左手坐标系，带有添加的显示屏幕

三维平移是二维平移的简单扩展：

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-37}$$

也就是说，$T(d_x, d_y, d_z)\cdot[x\ y\ z\ 1]^{\mathrm{T}}=[x+d_x\ \ y+d_y\ \ z+d_z\ 1]^{\mathrm{T}}$。

类似地，缩放扩展为：

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-38}$$

经核对，我们得到$S(s_x, s_y, s_z)\cdot[x\ y\ z\ 1]^{\mathrm{T}}=[s_x\cdot x\ s_y\cdot y\ s_z\cdot z\ 1]^{\mathrm{T}}$。

二维旋转式(5-26)仅仅是绕z轴进行的三维旋转，表示为

$$R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{5-39}$$

这很容易验证：对沿x轴的单位向量$[1\ 0\ 0\ 1]^{\mathrm{T}}$进行90°的旋转，得到沿$y$轴的单位向量$[0\ 1\ 0\ 1]^{\mathrm{T}}$。计算乘积

$$\begin{bmatrix}0 & -1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1\\ 0\\ 0\\ 1\end{bmatrix} \tag{5-40}$$

得到预测的结果$[0\ 1\ 0\ 1]^{\mathrm{T}}$。

绕x轴的旋转矩阵为

$$R_x(\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{5-41}$$

181 绕y轴的旋转矩阵为

$$R_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{5-42}$$

$R_z(\theta)$, $R_x(\theta)$和$R_y(\theta)$的左上角3×3子矩阵的列（和行）是互相垂直的单位向量，并且子矩阵的行列式为1，这意味着这三个矩阵是特殊正交阵，如在5.3节中所述。同样，由任意的旋转序列所形成的左上3×3子矩阵也是特殊正交阵。记住，正交变换保持距离和角度不变。

所有这些矩阵都有逆（逆矩阵）。T的逆通过对d_x, d_y和d_z取负得到；对于S，其逆通过将s_x，s_y和s_z替换为它们的倒数得到；三个旋转矩阵的逆通过对旋转角度取负值得到。

任何正交矩阵B的逆恰好是它的转置：$B^{-1}=B^{\mathrm{T}}$。事实上，转置不需要交换存储矩阵的数组的元素——只需要在存取数组时交换行列的下标即可。注意到这种求逆的方法与求R_x, R_y和R_z的逆对θ取负值的结果一致。

任意次的旋转、缩放和平移矩阵都可以乘到一起。其结果总是具有如下形式：

$$M=\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\ r_{21} & r_{22} & r_{23} & t_y\\ r_{31} & r_{32} & r_{33} & t_z\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{5-43}$$

正如二维中的情况那样，3×3左上子矩阵R给出合计的旋转和缩放，同时T给出随后总的平移。一些计算效率通过显式地执行如下变换获得

$$\begin{bmatrix}x'\\ y'\\ z'\end{bmatrix}=R\cdot\begin{bmatrix}x\\ y\\ z\end{bmatrix}+T \tag{5-44}$$

其中R和T是式(5-43)中的子矩阵。

与5.3节中的二维错切矩阵对应的是三维错切矩阵。(x, y)的错切为

$$SH_{xy}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-45}$$

对点$[x\ y\ z\ 1]^T$施加SH_{xy}，我们有$[x + sh_x \cdot z\ \ y + sh_y \cdot z\ \ z\ 1]^T$。沿$x$轴和$y$轴的错切具有相似的形式。

至此，我们一直集中于单个点的变换。由两个点定义的直线段通过变换其端点来实现整条直线段的变换。如果平面由三个点定义，则可以用相同的方法处理，但是通常平面用平面方程 182 来定义，方程的系数必须用不同的方法变换。令平面用平面方程的系数列向量$N = [A\ B\ C\ D]^T$表示，那么一个平面被定义为满足方程$N \cdot P = 0$的所有点，其中“·”是向量点积且$P = [x\ y\ z\ 1]^T$。这个点积得到熟悉的平面方程$Ax + By + Cz + D = 0$，该方程也可以表示为平面方程系数的行向量与列向量P的乘积：$N^T \cdot P = 0$。现在，假设我们通过某个矩阵M变换平面上所有的点P。为了使变换后的点保持$N^T \cdot P = 0$，我们应该用某个（待定）矩阵Q对N进行变换，得到方程$(Q \cdot N)^T \cdot M \cdot P = 0$。运用恒等式$(Q \cdot N)^T = N^T \cdot Q^T$，这种表示可以被依次重新写成$N^T \cdot Q^T \cdot M \cdot P = 0$。如果$Q^T \cdot M$是单位矩阵的倍数，则该等式成立。如果倍数是1，则有$Q^T = M^{-1}$，或$Q = (M^{-1})^T$。因此，用M变换以后的平面系数的列向量N'为

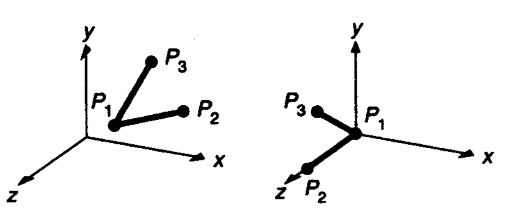
$$N' = (M^{-1})^T \cdot N \tag{5-46}$$

矩阵$(M^{-1})^T$一般不一定存在，因为M的行列式可能为0。如果M包括一个投影（我们可能想要研究平面上的透视投影的效果）。

如果仅仅变换平面的法线（例如，在第14章中讨论的明暗处理的计算），并且M仅包括平移、旋转和均匀缩放矩阵的合成，那么数学更加简单。式(5-46)中的N'可以被简化为$[A'\ B'\ C'\ 0]^T$。（如果有一个为0的W分量，该齐次点表示无穷远点，可以被看成一个方向。）

5.8 三维变换的合成

在本节中，我们用一个例子讨论如何合成三维变换矩阵，这个例子在6.5节还要用到。目标是将有向线段P_1P_2和P_1P_3从图5-18a的开始位置移到图5-18b的结束位置。所以，点P_1将被平移到原点，P_1P_2位于z轴的正方向上，P_1P_3位于y轴正向的(y, z)半平面上。直线的长度不受变换的影响。



a) 最初位置　　b) 最终位置

图5-18 将P_1, P_2和P_3从最初位置a)变换为它们的最终位置b)

这里给出两种完成所希望的变换的方法。第一种方法是将基本变换T, R_x, R_y和R_z合成起来。这种方法虽然有些烦琐，但是很容易用图例说明，而且理解这种方法会帮助我们建立对变换的理解。第二种方法利用5.7节所描述的正交矩阵的性质，可以更简明地解释但是要抽象一些。

要用基本变换实现，我们还要将一个复杂的问题分解为简单一些的子问题。在这种情况下，希望的变换可以用四步完成：

1) 将P_1平移到原点。
2) 绕y轴旋转使得直线P_1P_2位于（y, z）平面上。
3) 绕x轴旋转使得直线P_1P_2落在z轴上。
4) 绕z轴旋转使得直线P_1P_3位于（y, z）平面上。

第一步：将P₁平移到原点

平移变换为

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-47}$$

对P_1, P_2和P_3施加T变换得到

$$P_1' = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5-48}$$

$$P_2' = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} \tag{5-49}$$

$$P_3' = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} \tag{5-50}$$

第二步：绕y轴旋转

图5-19显示经过第一步后的P_1P_2，以及P_1P_2在（x, z）平面上的投影。旋转角度是$-(90-\theta)=\theta-90$。那么

$$\begin{aligned} \cos(\theta-90) = \sin\theta = \frac{z_2'}{D_1} = \frac{z_2 - z_1}{D_1} \\ \sin(\theta-90) = -\cos\theta = -\frac{x_2'}{D_1} = -\frac{x_2 - x_1}{D_1} \end{aligned} \tag{5-51}$$

其中

$$D_1 = \sqrt{(z_2')^2 + (x_2')^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2} \tag{5-52}$$

当这些值被代入式(5-42)中，我们得到

183~184

$$P_2'' = R_y(\theta-90) \cdot P_2' = [0 \;\; y_2 - y_1 \;\; D_1 \;\; 1]^{\mathrm{T}} \tag{5-53}$$

正如期望的那样，P_2''的x分量等于0，z分量等于长度D_1。

第三步：绕x轴旋转

图5-20显示经过第二步后的P_1P_2。旋转角度为ϕ，有

$$\cos\phi = \frac{z_2''}{D_2}, \quad \sin\phi = \frac{y_2''}{D_2} \tag{5-54}$$

其中$D_2 = |P_1''P_2''|$，为直线$P_1''P_2''$的长度。但是$P_1''P_2''$的长度与P_1P_2相等，因为旋转和平移变换保持长度不变，所以

$$D_2 = | P_1'' P_2'' | = | P_1 P_2 | = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{5-55}$$

第三步旋转的结果为

$$P_2''' = R_x(\phi)\cdot P_2'' = R_x(\phi)\cdot R_y(\theta-90)\cdot P_2'$$
$$= R_x(\phi)\cdot R_y(\theta-90)\cdot T\cdot P_2 = [0\quad 0\quad |P_1P_2|\quad 1]^{\mathrm{T}} \tag{5-56}$$

也就是说，P_1P_2与z轴的正向一致。

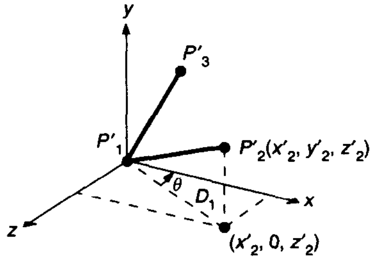


图5-19 绕y轴旋转：$P'_1P'_2$的投影（其长度为D_1）被旋转到z轴。角度θ表明绕y轴旋转的正方向：实际角度是$-(90-\theta)$

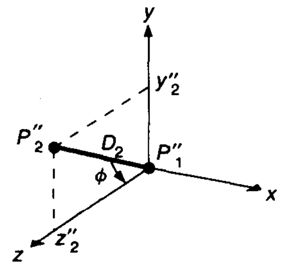


图5-20 绕x轴旋转：$P_1''P_2''$被以角度ϕ旋转到z轴，D_2是线段的长度。线段$P_1''P_3''$没有显示，因为它没有用于确定旋转的角度。两条线都按$R_x(\phi)$旋转

第四步：绕z轴旋转

图5-21显示经过第三步后的P_1P_2和P_1P_3，这时P_2'''在z轴上，P_3'''的位置为

$$P_3''' = [x_3'''\quad y_3'''\quad z_3'''\quad 1]^{\mathrm{T}} = R_{\mathrm{x}}(\phi)\cdot R_{\mathrm{y}}(\theta-90)\cdot T(-x_1,-y_1,-z_1)\cdot P_3 \tag{5-57}$$

旋转一个正的角度α，有

$$\cos\alpha = y_3'''/D_3,\quad \sin\alpha = x_3'''/D_3,\quad D_3=\sqrt{x_3'''^2+y_3'''^2} \tag{5-58}$$

第四步完成变换，结果如图5-18b所示。 185

合成矩阵

$$R_z(\alpha)\cdot R_x(\phi)\cdot R_y(\theta-90)\cdot T(-x_1,-y_1,-z_1) = R\cdot T \tag{5-59}$$

为所要求的变换，其中$R = R_z(\alpha)\cdot R_x(\phi)\cdot R_y(\theta-90)$。我们给读者留下一个问题，对$P_1$, P_2和P_3应用这个变换，以验证是否：P_1被平移到原点，P_2被平移到z轴正向，P_3被平移到y轴正向的(y, z)半平面。

第二种获得矩阵R的方法是利用5.3节中讨论的正交矩阵的特性。回忆一下R的单位行向量旋转成基本坐标轴。为了描述方便用x, y和z代替公式(5-43)中的第二个下标。

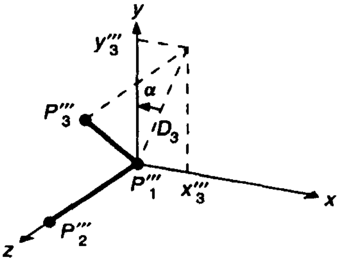


图5-21 绕z轴旋转：$P_1'P_3'$（其长度是D_3）的投影被旋转正α角到y轴，将线带到（y, z）平面。D_3是投影的长度

$$R = \begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} \\ r_{1_y} & r_{2_y} & r_{3_y} \\ r_{1_z} & r_{2_z} & r_{3_z} \end{bmatrix} \tag{5-60}$$

因为R_z是沿着P_1P_2的单位向量，P_1P_2将被旋转到z轴正向，所以

$$R_z = [r_{1_z}\quad r_{2_z}\quad r_{3_z}]^{\mathrm{T}} = \frac{P_1P_2}{|P_1P_2|} \tag{5-61}$$

另外，单位向量R_x与P_1、P_2和P_3构成的平面垂直，并且将被旋转到x轴正向，所以R_x一定是平面上两个向量规格化的叉积：

$$R_x = [r_{1_x} \quad r_{2_x} \quad r_{3_x}]^T = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|} \tag{5-62}$$

最后，

$$R_y = [r_{1_y} \ r_{2_y} \ r_{3_y}]^T = R_z \times R_x \tag{5-63}$$

将被旋转到y轴的正向。合成矩阵给出如下：

$$\begin{bmatrix} r_{1x} & r_{2_x} & r_{3_x} & 0 \\ r_{1y} & r_{2_y} & r_{3_y} & 0 \\ r_{1z} & r_{2_z} & r_{3_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_1, -y_1, -z_1) = R \cdot T \tag{5-64}$$

其中R和T与式(5-59)中的相同。图5-22显示单个的向量R_x、R_y和R_z。

现在考虑另外一个例子。图5-23显示一架定义在x_p，y_p和z_p坐标系中并以原点为中心的飞机。我们想要对飞机做一个变换，使得飞机朝向由向量DOF（飞行方向）指定的方向，以P为中心，并且不倾斜，如图5-24所示。达到这个结果的变换包括一个使飞机朝向合适方向的旋转，接着是从原点到P点的平移。为了找到旋转矩阵，我们仅需要确定图5-24中x_p, y_p和z_p轴的朝向，确保这些方向是归一化的，然后作为旋转矩阵的列向量使用它们。

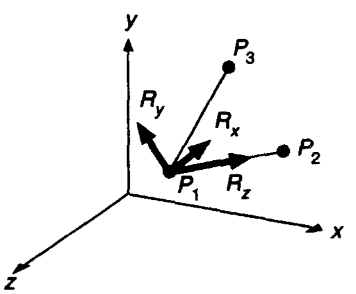


图5-22 单位向量R_x、R_y和R_z，它们被变换到主轴

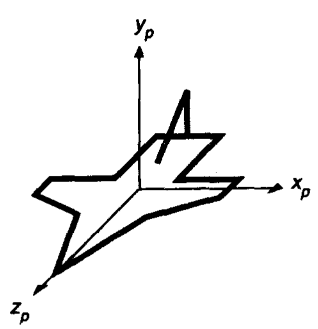


图5-23 (x_p, y_p, z_p)坐标系中的飞机

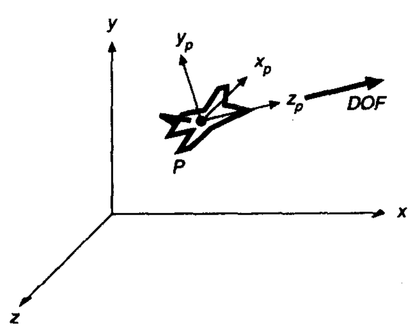


图5-24 图5-23的飞机定位在点P，并且头在DOF方向

z_p轴必须被变换到DOF的方向上，x_p轴则必须被变换为一个垂直DOF的水平向量——也就是说，在$y \times DOF$的方向上，即y和DOF的叉积。y_p方向由$z_p \times x_p = DOF \times (y \times DOF)$给出，即$z_p$和$x_p$的叉积；因此，旋转矩阵的三个列向量分别为归一化的向量$|y \times DOF|$、$|DOF \times (y \times DOF)|$和$|DOF|$：

$$R = \begin{bmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-65}$$

DOF与y轴同向是一种退化的情况，因为此时的水平向量有无限种可能。这种退化也被反映到代数上，因为$y \times DOF$和$DOF \times (y \times DOF)$等于0。在这种特殊情况下，$R$不再是旋转矩阵。

5.9 坐标系的变换

我们一直在讨论当两个点集在同一个坐标系中时将属于一个物体的点集变换为另一个点集。用这种方法，坐标系保持不变而物体相对于坐标原点被变换。考虑变换的另一种可选择并且等价的方式是坐标系的改变。这种观点在多个物体被结合时十分有用，其中每个物体都被定义在自己的局部坐标系中，而我们希望在一个全局坐标系中表示这些物体的坐标。这种情况在第7章中将要出现。186～187

我们定义$M_{i\leftarrow j}$为一个变换，该变换将点在坐标系j中的表示转换为在坐标系i中的表示。

我们再定义$P^{(i)}$为一个点在坐标系i中的表示，$P^{(j)}$为点在坐标系j中的表示，$P^{(k)}$为点在坐标系k中的表示，那么，

$$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)} \text{ and } P^{(j)} = M_{j\leftarrow k} \cdot P^{(k)} \tag{5-66}$$

替换$P^{(j)}$，我们得到

$$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \cdot P^{(k)} = M_{i\leftarrow k} \cdot P^{(k)} \tag{5-67}$$

所以

$$M_{i\leftarrow k} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \tag{5-68}$$

图5-25显示四个不同的坐标系。通过观察我们看到从坐标系2到坐标系1的变换为$M_{1\leftarrow 2} = T(4, 2)$。相似地，$M_{2\leftarrow 3} = T(2, 3) \cdot S(0.5, 0.5)$，$M_{3\leftarrow 4} = T(6.7, 1.8) \cdot R(45°)$。那么$M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(0.5, 0.5)$。这个图也显示了一个点在坐标系1到坐标系4中的坐标分别为$P^{(1)} = (10, 8)$，$P^{(2)} = (6, 6)$，$P^{(3)} = (8, 6)$和$P^{(4)} = (4, 2)$。很容易验证$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$，当$1 \leq i, j \leq 4$时。

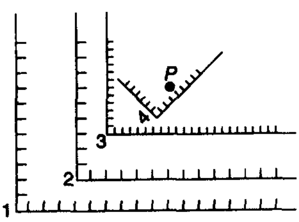


图5-25 点P和坐标系1, 2, 3, 4

我们也注意到$M_{i\leftarrow j} = M_{j\leftarrow i}^{-1}$。所以，$M_{2\leftarrow 1} = M_{1\leftarrow 2}^{-1} = T(-4, -2)$。因为$M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3}$，所以$M_{1\leftarrow 3}^{-1} = M_{2\leftarrow 3}^{-1} \cdot M_{1\leftarrow 2}^{-1} = M_{3\leftarrow 2} \cdot M_{2\leftarrow 1}$。

在5.7节中，我们讨论过左手坐标系和右手坐标系。将点在左手坐标系和右手坐标系之间转换的矩阵是它本身的逆，表示为

$$M_{R\leftarrow L} = M_{L\leftarrow R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-69}$$

在前面几节中使用的方法——在世界坐标系中定义所有的物体，然后将它们变换到希望的位置——暗含着有些不现实的概念，即所有的物体初始都一个接一个地定义在同一个世界坐标系下。一种更加自然的想法是认为每个物体定义在自己的坐标系中，然后通过在新的世界坐标系中重定义它的坐标来进行缩放、旋转和平移变换。在第二种观点中，我们会自然地想到撕开的纸片，每张纸片上有一个物体，而纸片在世界坐标系的平面上被缩小或拉伸、旋转或平移。我们当然也可以想像平面相对于每张纸片被缩小或拉伸、倾斜或者滑动。数学上说，所有这些观点都是等价的。

考虑简单的情况，将图5-10中定义房子的点集平移到原点。这个变换是$T(-x_1, -y_1)$。在图5-26中标记两个坐标系，我们看到将坐标系1映射成坐标系2的变换（即$M_{2\leftarrow 1}$）为$T(x_1, y_1)$，恰好是$T(-x_1, -y_1)^{-1}$。确实，通常的规则是在单个坐标系中对点集进行的变换恰好是对应的改变

点被表示的坐标系的变换的逆。这种关系可以从图5-27中看出，也可以直接从图5-11中得到。在单个坐标系下表示的点的变换为

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \tag{5-32}$$

在图5-27中，坐标系变换正是

$$\begin{aligned} M_{5\leftarrow 1} &= M_{5\leftarrow 4} M_{4\leftarrow 3} M_{3\leftarrow 2} M_{2\leftarrow 1} \\ &= (T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1))^{-1} \\ &= T(x_1, y_1) \cdot S(s_x{}^{-1}, s_y{}^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \end{aligned} \tag{5-70}$$

所以

$$P^{(5)} = M_{5\leftarrow 1} P^{(1)} = T(x_1, y_1) \cdot S(s_x{}^{-1}, s_y{}^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \cdot P^{(1)} \tag{5-71}$$

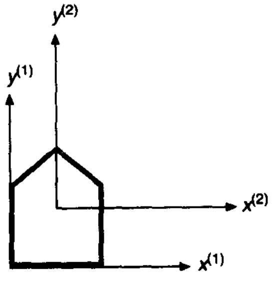


图5-26　房子和两个坐标系。房子上点的坐标被在两个坐标系中表示

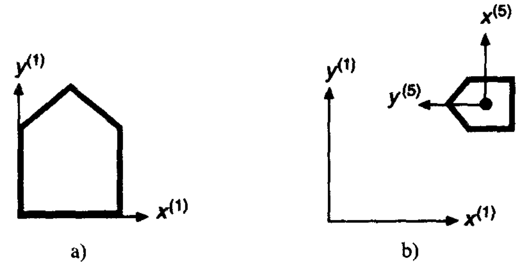


图5-27　原始房子a)在其坐标系中和在其坐标系中被变换的房子b)相对于原始坐标系

与改变坐标系相关的一个重要问题是我们如何改变变换。假设$Q^{(j)}$是坐标系j中的变换。举例说明，它可能是从前面几节中提取出的合成变换。假设我们想要找到一个在坐标系i中的变换$Q^{(i)}$，该变换将被应用到坐标系i中的点$P^{(i)}$上，并产生与$Q^{(j)}$被应用到坐标系j中相对应的点$P^{(j)}$相同的结果。这个等式被表示为$Q^{(i)} \cdot P^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$。将$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$代入，等式变为$Q^{(i)} \cdot M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$。经过简化，我们有$Q^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot M_{i\leftarrow j}{}^{-1}$。

当用户在后面的局部坐标系中指定了子物体的额外信息时，改变坐标系的观点将十分有用。例如，如果在图5-28中三轮车的前轮绕z_{wh}坐标旋转，所有的轮子必须被恰当地旋转，而且我们需要知道三轮车作为一个整体如何在世界坐标系中移动。这个问题非常复杂，因为几个连续的

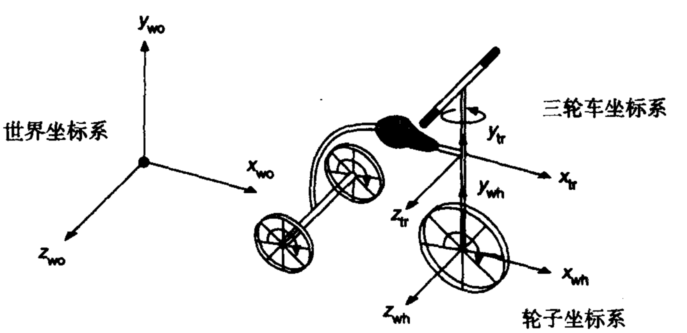


图5-28　带有三个坐标系的三轮车

坐标系变换同时发生。首先，三轮车和前轮坐标系在世界坐标系中具有初始的位置。当三轮车向前移动时，前轮绕车轮坐标系的z轴旋转，与此同时车轮坐标系和三轮车坐标系相对于世界坐标系移动。车轮坐标系和三轮车坐标系通过在x和z方向上随时间变化的平移加上绕z轴的旋转，与世界坐标系相关联。当手把被旋转时，车轮坐标系与三轮车坐标系则通过一个绕z轴随时间变化的旋转相互关联。（三轮车坐标系固定在车架上，而不是在手把上。）

为了使问题更加简单，我们假设车轮和三轮车坐标系的轴平行于世界坐标系的轴，而且车轮在与世界坐标系x轴平行的一条直线上运动。当车轮旋转α角度时，车轮上的一点P旋转路程αr，其中r是车轮的半径。因为车轮在地面之上，三轮车向前移动αr单位。所以，车轮上的边缘点P相对于初始车轮坐标系移动和旋转，产生一个连续经过平移距离αr和旋转角度α的最终效果。因此，在原始车轮坐标系中新的坐标P'为

$$P'^{(\mathrm{wh})} = T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(\mathrm{wh})} \tag{5-72}$$

并且它在新的（经过平移）车轮坐标系中的坐标仅由旋转给出

$$P'^{(\mathrm{wh}')} = R_z(\alpha) \cdot P^{(\mathrm{wh})} \tag{5-73}$$

为了在世界坐标系中找到点$P^{(\mathrm{wo})}$和$P'^{(\mathrm{wo})}$，我们将车轮坐标系变换到世界坐标系：

$$P^{(\mathrm{wo})} = M_{\mathrm{wo}\leftarrow\mathrm{wh}} \cdot P^{(\mathrm{wh})} = M_{\mathrm{wo}\leftarrow\mathrm{tr}} \cdot M_{\mathrm{tr}\leftarrow\mathrm{wh}} \cdot P^{(\mathrm{wh})} \tag{5-74}$$

$M_{\mathrm{wo}\leftarrow\mathrm{tr}}$和$M_{\mathrm{tr}\leftarrow\mathrm{wh}}$是平移矩阵，由三轮车和车轮的初始位置给出。

$P'^{(\mathrm{wo})}$可以用式(5-72)和式(5-74)求得：

$$P'^{(\mathrm{wo})} = M_{\mathrm{wo}\leftarrow\mathrm{wh}} \cdot P'^{(\mathrm{wh})} = M_{\mathrm{wo}\leftarrow\mathrm{wh}} \cdot T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(\mathrm{wh})} \tag{5-75}$$

另外，我们认识到通过车轮坐标系的平移$M_{\mathrm{wo}\leftarrow\mathrm{wh}}$被变换为$M_{\mathrm{wo}\leftarrow\mathrm{wh}'}$，以一种不同的方式得到与式(5-75)相同的结果：

$$P'^{(\mathrm{wo})} = M_{\mathrm{wo}\leftarrow\mathrm{wh}'} \cdot P'^{(\mathrm{wh}')} = (M_{\mathrm{wo}\leftarrow\mathrm{wh}} \cdot M_{wh\leftarrow\mathrm{wh}'}) \cdot (R_z(\alpha) \cdot P^{(\mathrm{wh})}) \tag{5-76}$$ 188~190

因此，一般而言，通过从三轮车部件的运动方程应用合适的变换，我们可以从以前的值求出新的$M_{\mathrm{wo}\leftarrow\mathrm{wh}'}$和$M_{\mathrm{tr}'\leftarrow\mathrm{wh}'}$。我们再对局部坐标系中更新后的点应用更新后的变换，得到世界坐标系中等价的点。

习题

5.1 证明：通过变换一条直线的端点，然后在变换后的端点之间构造新的直线段，我们可以实现对直线段的变换。

5.2 证明：连续的二维旋转是加法：$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$。

5.3 证明：如果$s_x = s_y$或$\theta = n\pi$，其中n为整数，二维旋转和缩放可交换，否则不可交换。

5.4 对点P_1、P_2和P_3应用5.8节中的变换，以验证这些点与预想的一样变换。

5.5 重做5.8节中的工作，假设$|P_1P_2| = 1$，$|P_1P_3| = 1$，并且P_1P_2 和P_1P_3的方向余弦已知（直线的方向余弦是直线和x、y和z轴之间的夹角的余弦）。对于一条从原点到(x, y, z)的直线，方向余弦等于$(x/d, y/d, z/d)$，其中d是直线的长度。

5.6 证明：式(5-59)与式(5-64)等价。

5.7 已知一个单位立方体，其中一个角点在(0, 0, 0)，另一个对角点在(1, 1, 1)，导出绕主对角线（从(0, 0, 0)到(1, 1, 1)）以逆时针方向旋转θ 角所必需的变换。逆时针方向是沿着对角线向原点看。

5.8 假设窗口的底边被从x轴旋转θ角，就像在Core System[GSPC79]。什么是窗口到视口的映射？通过对窗口的每个角点应用变换来验证你的答案，确认这些角点被变换到合适的视口角点。

5.9 考虑一条从右手坐标系的原点到点$P(x, y, z)$的直线。用三种不同的方式找到将直线旋转到z轴正方向所需的变换矩阵，并用代数操作显示每一种情况中P确实到达z轴。对于每一种方法，计算旋转角度的正弦和余弦。

a. 绕y轴旋转到(y, z)平面，然后绕x轴旋转到z轴。

b. 绕z轴旋转到(x, z)平面，然后绕y轴旋转到z轴。

5.10 将一个物体在方向余弦为（α, β, γ）的方向上以缩放因子S进行缩放。导出变换矩阵。

5.11 找到绕任意方向旋转θ角的4×4变换矩阵，该方向由方向向量$U = (u_x, u_y, u_z)$给出。通过合成变换矩阵完成这个任务，首先用一个旋转变换$R_z(\theta)$将U旋转到z轴（记为M），然后

191 用M^{-1}合成这个结果。结果应该是

$$\begin{bmatrix} u_x^2 + \cos\theta(1 - u_x^2) & u_x u_y(1 - \cos\theta) - u_z \sin\theta & u_z u_x(1 - \cos\theta) + u_y \sin\theta & 0 \\ u_x u_y(1 - \cos\theta) + u_z \sin\theta & u_y^2 + \cos\theta(1 - u_y^2) & u_y u_z(1 - \cos\theta) - u_x \sin\theta & 0 \\ u_z u_x(1 - \cos\theta) - u_y \sin\theta & u_y u_z(1 - \cos\theta) + u_x \sin\theta & u_z^2 + \cos\theta(1 - u_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-77}$$

验证：如果U是一个主轴，矩阵变为R_x、R_y或R_z。基于向量操作的推导参见[FAUX79]。

192 注意，同时对U和θ取负值结果保持不变。解释其原因。

第6章 三维空间的观察

在三维空间中的观察过程根本就比在二维空间中的观察过程复杂得多。对于二维情况，我们仅需要在二维空间指定一个窗口并在二维观察表面给定一个视口。从概念上讲，可以使用窗口对世界的物体进行裁剪，然后变换到视口进行显示。三维观察过程的额外的复杂性一部分是由被添加的维引起的，还有一部分是由显示设备仅是二维的这一事实引起的。虽然三维观察过程刚开始看起来好像很复杂，但当我们将它看作为一系列容易理解的步骤时，就不会烦恼了。其中许多步骤我们在前几章中已有所准备了。因此，我们开始用三维观察过程的一个大纲来帮助引导读者阅读整章。

6.1 人造照相机及三维观察步骤

建立三维景象的一个有用隐喻是使用一种**人造（假想）照相机**的概念，图6-1说明了这一概念。我们想像能够移动照相机到任意位置，用我们希望的任意方法来定向它，而且可以用快门对三维物体（这里是高速游艇）获取二维图像的快照。按我们的指令，它能成为一个移动图片的照相机，使得我们能够创建一系列活动图片，以便按不同的方向和放大倍数来展示物体。当然，照相机实际上是在显示屏产生图像的一个计算机程序，而物体是由一组点、线、面组成的三维数据集。图6-1也展示了照相机和三维物体，每一个都有它们自己的坐标系统：照相机为u, v, n；物体为x, y, z。在本章的后面部分，我们将讨论这些坐标系统的意义。这里我们注意到，它们提供了一种重要的表示独立性。 193

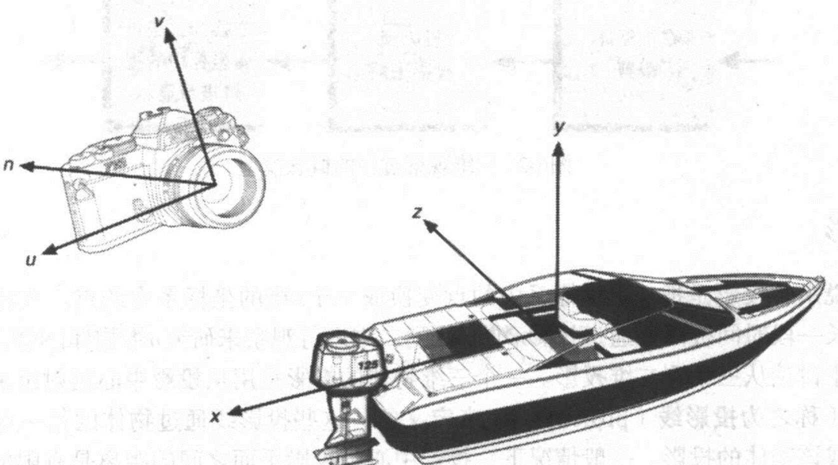


图6-1 对三维物体照相的一个人造照相机

虽然人造照相机是一个有用的概念，但从产生一个图像来看，它比只按一下按钮要多一点麻烦。事实上创建我们的“照片”需实现一系列步骤，描述如下：

- 投影类型的说明。我们通过引入**投影**来解决三维物体和二维显示之间的配合不当，投影把三维物体变换到一个二维投影平面上。本章的许多内容将讨论到投影：什么是投影，

投影的数学问题是什么，投影是如何被用于PHIGS[ANSI88]，一个最近的图形子程序软件包。我们集中在两类最重要的投影上：**透视投影**和**平行正投影**。投影的使用还将在第7章中进一步讨论。

- *观察参数的说明*。期望的投影类型一旦确定，必须说明在该投影类型下我们观察三维数据集或绘制场景的条件。当三维数据集的世界坐标给定后，这类条件的信息包括观察者眼睛的位置、观察平面（最终显示投影的平面）的位置。我们将使用两个坐标系统：一个是景物的坐标系统，另一个我们称其为**观察**或**眼睛坐标系统**。通过所有这些参数或者
194 它们的变动，我们能够实现所希望景物的任何表示，包括有意义时观察其内部。
- *在三维空间中裁剪*。就像我们必须限制一个二维景象在我们指定窗口的边界内展开显示那样，对于一个三维景象我们也必须精选出其中用于最终显示的那一部分。事实上为清晰起见，我们可以忽略在我们身后的或者很远距离的那部分景象。这个活动要求我们按视见体进行裁剪——这是一个比我们已经讨论过的算法表示的更为复杂的过程。由于视见体具有潜在的很宽的变化范围，我们将着力定义一个规范的**视见体**——针对它，我们可以有效地应用一个标准的裁剪算法。
- *投影和显示*。最后，视见体在投影平面（称为**窗口**）上投影的内容将变换（映射）到显示器的视口内。

图6-2表示了三维观察过程概念模型中的主要步骤，这是提供给许多三维图形子程序包用户的模型。和二维观察的情况一样，可以采用各种各样的策略实现一个观察过程。这些策略不需要与概念模型一样，只要结果是模型定义的就可以了。一个典型的线框图的实现策略在6.6节描述。对于实现可见面确定和明暗处理的图形系统，应用了有一点不同的流水线，这些将在第14章讨论。

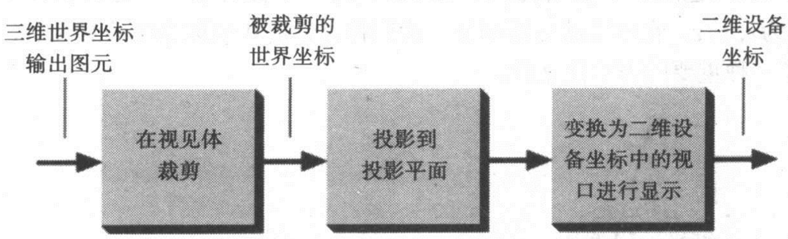


图6-2 三维观察过程的概念模型

6.2 投影

一般说来，投影是把 *n*维坐标系中的点变换成小于*n*维的坐标系中的点。实际上，计算机图形学很长一段时间被用来通过投影*n*维物体到二维进行观察来研究*n*维物体[NOLL67]。这里，我们将限于讨论从三维到二维投影。一个三维物体的投影是用从**投影中心**发射出来的许多直的
195 投影射线（称之为**投影线**（projector））来定义的，这些投影线通过物体的每一点和**投影平面**相交，形成该物体的投影。一般情况下，投影中心与投影平面之间的距离是有限的。但对于某些投影类型，认为投影中心在无穷远处更为方便，在6.2.1节中我们将进一步探讨这一概念。图6-3显示同一条直线的两个不同的投影。幸运的是，一条直线段的投影本身仍是一条直线段，因此，实际上只需对直线段的端点做投影变换。

我们在这里处理的这类投影被称为**平面几何投影**，因为投影是到平面上而不是到曲面上，并且投影线是直线而不是曲线。许多制图学方面的投影可能或者是非平面的或者是非几何的。

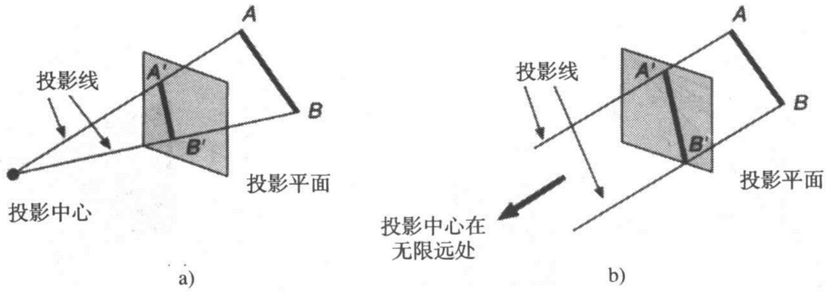


图6-3 同一条直线的两个不同的投影。a) 直线AB和它的透视投影$A'B'$，b) 直线AB和它的平行投影$A'B'$。投影线AA'和BB'是平行的

平面几何投影（此后简称为**投影**）可以分为两种基本类型：**透视投影**和**平行投影**。它们的区别在于投影中心与投影平面之间的关系不同。如果投影中心到投影平面之间的距离是有限的，那么投影是透视投影；随着投影中心越移越远，穿过任何特定物体的投影线越来越接近平行。图6-3说明了这两种情况。平行投影之所以这样命名，是因为投影中心到投影平面之间的距离是无限的，所以投影线是互相平行的。在定义透视投影的时候，我们显式地给定它的**投影中心**；对于平行投影，我们给出它的**投影方向**。投影中心是一个点，具有形式（$x, y, z, 1$)的齐次坐标。由于投影方向是一个向量（也就是点间坐标之差），可以通过两个点相减获得：$d = (x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)$。所以**方向**和**无穷远点**以一种自然的方式对应。在这一约束下，中心是一个无限点的透视投影成为平行投影。

透视投影的视觉效果类似于照相系统和人的视觉系统，称为**透视缩小效应**（perspective foreshortening）：一个物体的透视投影的大小与物体到投影中心的距离成反比。因此，尽管物体的透视投影倾向于看起来真实，但这对于记录物体的精确形状和尺寸并不特别有用，因为不能从物体的投影获得距离，而且仅保留那些与投影平面相平行的物体表面的角度，平行线的投影一般并不是平行的。 196

平行投影产生的视图真实性较差，这是因为虽然沿着每一个坐标轴都有不同的固定透视缩小系数，但仍然不具备透视缩小效应。平行投影可以被用于精确的测量，并且平行线确实保持平行。与透视投影相同，也保留了那些与投影平面平行的物体各表面的角度。

不同类型的透视投影和平行投影在Carlbom和Paciorek的综述论文中被详细讨论和说明[CARL78]。在6.2.1节和6.2.2节里我们总结被更广泛使用的投影的基本定义和特征；然后转向6.3节以理解对PHIGS各种投影怎样被实际定义。

6.2.1 透视投影

任何一束不平行于投影平面的平行线的透视投影将会聚成一个点，称为**灭点**（vanishing point）。在三维空间中，平行线仅在无限远处相聚，于是灭点可被看成无限远处一个点的投影。当然也存在着一个灭点的无穷远点，它是每一条直线指向的方向的无穷远。

如果直线的集合平行于三个主轴之一，这时的灭点称为**轴灭点**（axis vanishing point）。至多存在三个这样的轴灭点，对应于投影平面切割的主轴的数目。例如，如果投影平面仅切割z轴（因此，z轴是投影平面的法线），则只在z轴上有一个主灭点，因为此时平行于x或y坐标轴的直线也平行于投影平面，所以没有灭点。

透视投影是按主灭点的数量来分类的，所以也取决于投影平面切割坐标轴的数目。图6-4显示一个立方体的两种不同的一点透视投影。很清楚的是，它们是一点投影，因为平行x轴和y轴的直线并不会聚；仅有平行于z轴的直线才会聚于一点。图6-5显示具有一些投影线的一点透视的结构，且其投影平面仅切割z轴。

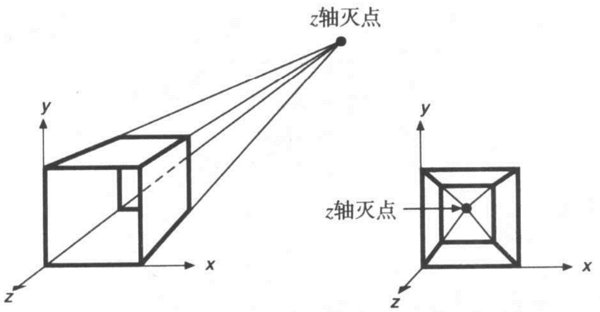


图6-4 立方体投影到切割z轴的平面的一点透视投影，显示垂直于投影平面的直线的灭点

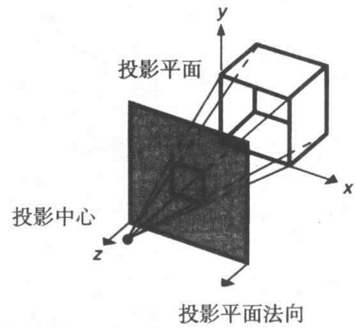


图6-5 立方体投影到切割 z 轴的平面的一点透视投影的结构。投影平面法线平行于z轴。（摘自[CARL78], Association for Computing Machinery, Inc.；授权使用。）

图6-6显示两点透视的结构。注意，在投影中平行于y轴的直线并不会聚于一点。两点透视通常应用于建筑、工程、工业设计和广告绘画等方面。三点透视应用得不多，因为它们并不比两点透视添加更多的真实性。

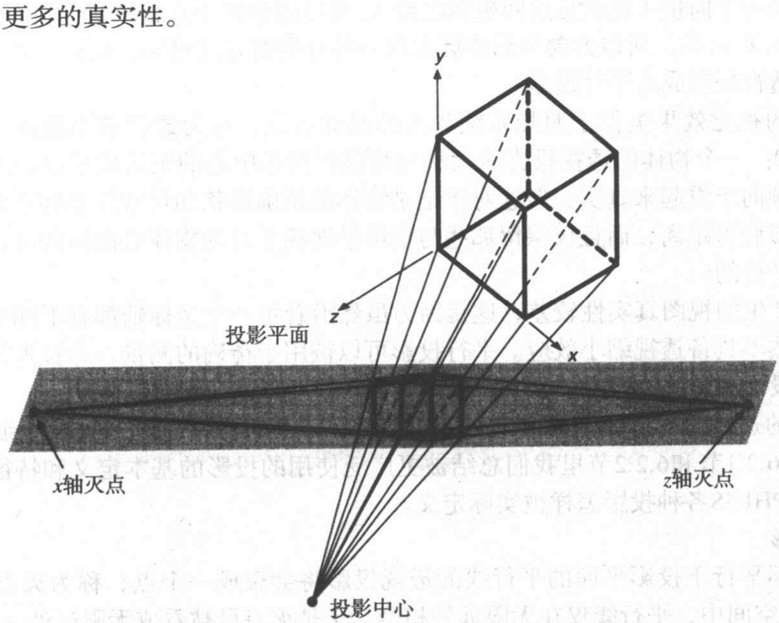


图6-6 立方体的两点透视投影。投影平面切割x轴和z轴

6.2.2 平行投影

平行投影根据投影方向和投影平面法线之间的关系分为两种类型。在**正平行投影**（orthographic parallel projection）中，这些方向是相同的（或彼此相反），所以投影方向是投影平面的法向。对于**斜平行投影**（oblique parallel projection），这些方向是不同的。

最常见的正投影类型是：**前视图**（主视图）投影，**顶视图**（俯视图，也叫**平面视图**）投影和**侧视图**投影。在所有这些投影中，投影平面垂直于一根主轴，所以该轴即为投影方向。图6-7显示这三种投影的结构，它们通常被用来在工程制图中绘制机械部件、机械零件的装配和建筑

物，因为用这些图可以测量出距离和角度。但是由于每一种投影仅描绘出物体的一个面，很难从中推导出被投影物体的三维性质，即使对于同一物体的几个投影同时研究也是如此。

轴测正投影（axonometric orthographic projection）使用的投影平面一般不垂直于某一个主轴，所以可以同时显示一个物体的几个面。在这一方面，就像透视投影一样，但是不同的是它的透视缩小系数是一致的而与到投影中心的距离无关。线的平行性保持不变，但是角度不是这样的，同时沿每根主轴，其距离是可度量的（一般具有不同的缩放因子）。

等轴测投影（isometric projection）是最常用的轴测投影。它的投影平面的法线（因此也是投影方向）与每根主轴的夹角相等。如果投影平面的法向量是(d_x, d_y, d_z),那么我们要求$|d_x| = |d_y| = |d_z|$或者$\pm d_x = \pm d_y = \pm d_z$。这样就有8个方向满足这样的条件（在每一个卦限中有一个方向）。图6-8显示一个沿方向(1，－1，－1)的等轴测投影结构。

等轴测投影具有有用的性质，即沿三个主轴方向具有相等的透视缩小系数，允许沿着三个轴向的量度具有相同的比例（因此才有等轴测的名字：iso代表相等，metric代表度量）。另外，主轴的投影使得120°的角与另一个角相等。

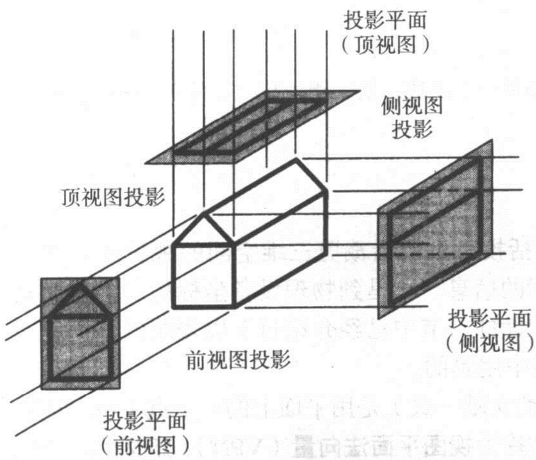


图6-7 三个正投影的结构

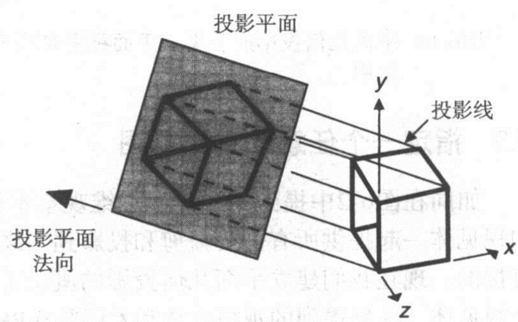


图6-8 一个单位立方体的等轴测投影的结构。（摘自[CARL78]，Association for Computing Machinery，Inc.；授权使用。）

斜投影，第二类平行投影，与正投影的不同之处在于投影平面的法向和投影方向不同。斜投影将轴测投影和前、顶、侧正投影的性质结合起来：投影平面垂直于一个主轴，于是平行于投影平面的物体表面的投影就可以进行角度和距离的测量。物体的另外一些表面也投影，允许沿着主轴测量其距离，但不是角度。斜投影在本书中被广泛（虽然不惟一）使用，这是由于它们本身的属性而且它们易于绘制。图6-9显示斜投影的结构。注意，投影平面法向和投影方向不一致。在[FOLE90]中描述了几种类型的斜投影。

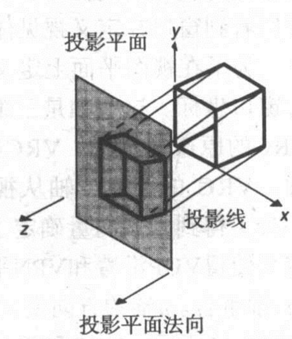


图6-9 斜投影的结构。（摘自[CARL78]，Association for Computing Machinery，Inc.；授权使用。）

图6-10显示不同类型的投影之间的逻辑关系。连接它们的共同线索是它们包含一个投影平面和透视投影的一个投影中心或者平行投影的投影方向。通过将投影中心看成由指向某个参考点的投影中心的方向与到该参考点的距离所定义，我们可以更进一步地统一平行投影和透视投影。当这个距离增加到无限，投影就成为

197
~
200

一个平行投影。因此，我们可以说统一这些投影的共同线索是它们本身包含一个投影平面，一个指向投影中心的方向和一个到投影中心的距离。在6.3节中，我们考虑怎样混合这些不同类型的投影到三维观察过程中。

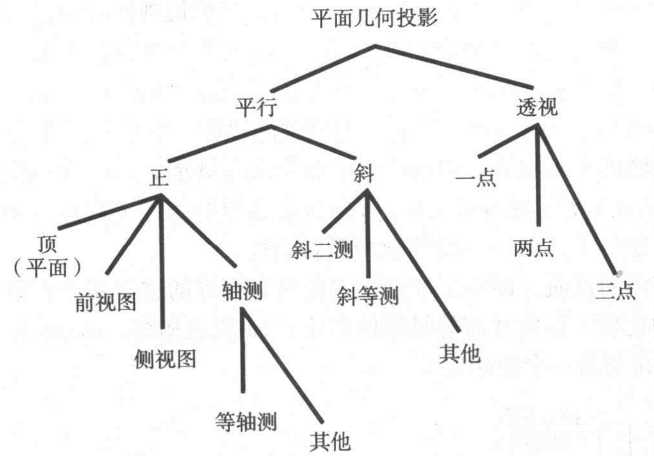


图6-10 平面几何投影的子类。**平面视图**是顶视图的另一个说法。**前**和**侧**经常是不带术语**视图**被使用

6.3 指定一个任意的三维视图

如同在图6-2中提到的那样，三维观察不仅包括投影也包括裁剪三维空间的视见体。投影和视见体一起提供所有需要裁剪和投影到二维空间的信息。于是到物理设备坐标的二维变换是直接的。现在我们建立平面几何投影的概念（在前面的一节中已经介绍过）以表明怎样确定一个视见体。这里提到的观察方法和术语是在PHIGS中用到的。

投影平面（以后称为**视图平面**以保持与图形学的文献一致）是用平面上的一个点（该点被称

201 为**视图参考点**（VRP））和该平面的一个法向量（被称为**视图平面法向量**（VPN））定义的。相对于被投影的物体，视图平面可以放在任何位置：可以穿过该物体，也可以放在物体的前面或后面。

给定视图平面，我们需要一个在视图平面上的窗口。窗口的作用与二维窗口的作用类似：它的内容被映射到视口，而投影到窗口外的视图平面上三维空间的任何部分是不显示的。我们可以看到窗口在定义视见体的时候也起到重要的作用。

为了在视图平面上定义一个窗口，我们需要一些方法来指定沿两个垂直坐标轴的最小和最大窗口坐标。这些轴是三维**观察参考坐标**（viewing-reference coordinate，VRC）系的一部分。VRC的原点是VRP。VRC的一个轴是VPN，该轴被称作n轴。VRC的第二个轴从**视图上方向量**（view-up vector，VUP）得到，该向量确定了视图平面上v轴的方向。v轴被定义使得VUP沿着和VPN平行的方向在视图平面上的投影与v轴重合。u轴的方向定义使u, v和n构成一个右手坐标系，如图6-11所示。VRP和两个方向向量VPN和VUP定义于右手世界坐标系中（某些图形程序包将y轴作为VUP，但这样限制制太大，如果VPN平行于y轴，将无法定义。在此情况下VUP会是无定义的）。

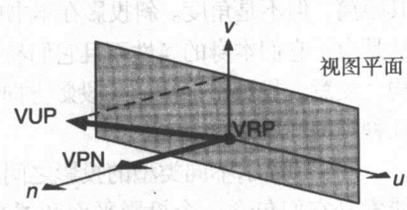


图6-11 视图平面由VPN和VRP定义；v轴由沿着VPN到视图平面的VUP的投影定义。u轴与VPN和v轴组成右手观察参考坐标系

定义了VRC系之后，就可以定义窗口的最小和最大的u值和v值，如图6-12所示。该图说明窗口不需要对称于视图参考点，并且清楚地显示了窗口的中心CW。

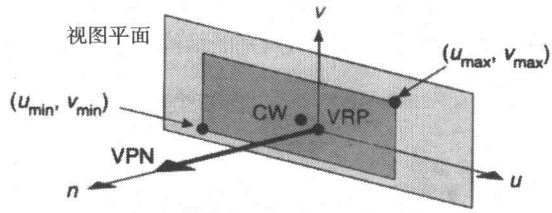


图6-12 观察参考坐标系统（VRC）是一个由u、v和n轴组成的右手系统。n轴总是VPN。CW是窗口中心

投影中心和投影方向（DOP）由**投影参考点**（projection reference point, PRP）和一个投影类型指针定义。如果投影类型是透视投影，则PRP是投影中心。如果投影类型是平行投影，则DOP从PRP指向CW。CW一般不是VRP，甚至不需要在窗口边界以内。

PRP在VRC系统中定义，而不是在世界坐标系中。因此，PRP 相对于VRP的位置并不随着VUP或VRP的移动而改变。这样的好处是程序员可以指定要求的投影方向，例如，使用斜等测投影，然后改变VPN和VUP(因此改变了VRC)，而不必要重新计算需要保持预期投影的PRP。另一方面，移动PRP来获得物体的不同视图会更困难一些。

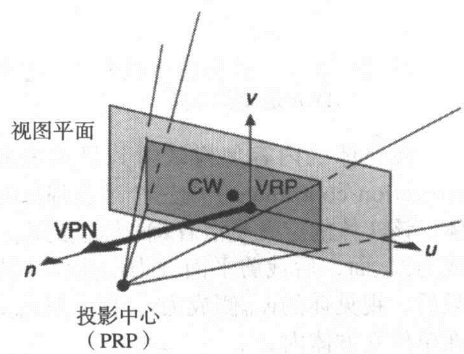


图6-13 透视投影的半无限四棱锥视见体。CW是窗口中心

视见体为世界空间中将被裁剪出来并投影到视图平面的那一部分定出边界。对透视投影，视见体是一个半无限的四棱锥形，其顶点是在PRP，边则穿过窗口的拐角。图6-13显示一个透视投影的视见体。

投影中心后面的位置不包含在视见体中，因此不会被投影。当然，在现实中，我们的眼睛看到一个不规则形状的圆锥形的视见体。但是，一个四棱锥形的视见体是数学上更易处理的，且与矩形视口的概念相一致。

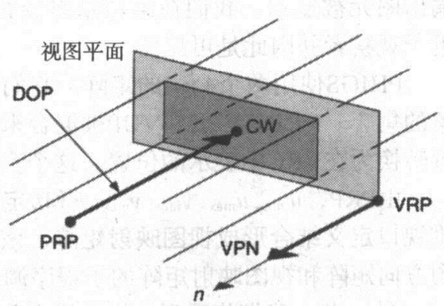


图6-14 平行正投影的无限平行管道视见体。VPN和投影的方向（DOP）是平行的。DOP是从PRP到CW的向量，且平行于VPN

对于平行投影，视见体是一个无限长的平行管道，它的各边都平行于投影方向，该方向是从PRP到窗口中心的方向。图6-14显示平行投影视见体和它们与视图平面、窗口和PRP的关系。

有时为了限制投影到视图平面上的输出图元的数量，我们可能需要视见体是有限的。图6-15和图6-16显示怎样用一个**前裁剪平面**（front clipping plane）和一个**后裁剪平面**（back clipping plane）使视见体成为有限的。这些平面（有时称为**前截面**（hither plane）和**后截面**（yon plane））均平行于视图平面；它们的法线是VPN。这些平面由相对于视图参考点和沿着VPN的称为**前距离**（F）或**后距离**（B）的带符号的量来定义，正向取在VPN方向。对于正的视见体，前距离必须在代数上大于后距离。

以这种方式限制视见体是有用的，可以消除无关物体同时允许用户关注于空间的一个特别的部分。前后距离的动态修改可以给观察者在物体的不同部分之间的空间关系上一种好的感觉（当这些部分在视野中出现和从视野中消失时，参见第12章）。对于透视投影，则有另外一个动机。恰好处在投影中心距离上的一个物体在投影平面上成为不可区别形状的斑点。在将这样的物体显示到绘图仪

上时，笔可能穿破纸；在向量显示器上，CRT荧光物质可能被电子束烧坏；在胶片记录器上，光线的高度集中导致出现一个模糊的白色区域。同时，非常靠近投影中心的物体可像许多不连接的采集棍一样扩展过窗口，没有可辨别的结构。适当地指定视见体可以消除这些问题。

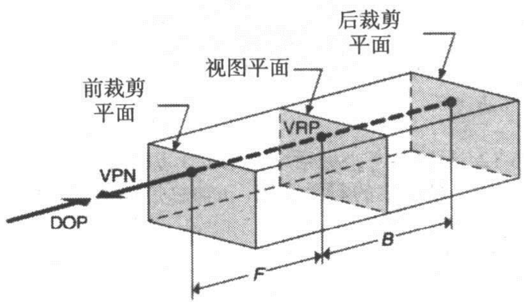


图6-15 正平行投影的被截断的视见体。DOP是投影方向

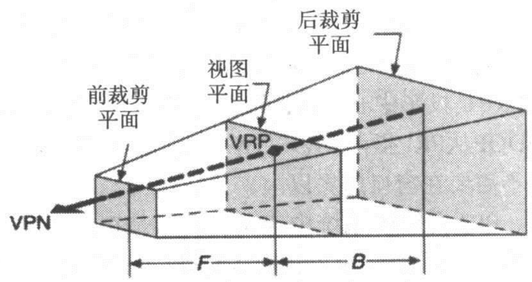


图6-16 透视投影的截断的视见体

视见体的内容怎样映射到显示表面上呢？首先，考虑在**规格化投影坐标**（normalized projection coordinate）的三个维度都从0到1扩展的单位立方体。视见体被转换为NPC 的矩形实体，该实体的定义是沿着x轴从x_{min}到x_{max}，沿着y轴从y_{min}到y_{max},沿着z轴从z_{min}到z_{max}。前裁剪平面成为z_{max}面，后裁剪平面成为z_{min}面。同样地，视见体的u_{min}侧成为x_{min}面，且u_{max}侧成为x_{max}面。最后，视见体的v_{min}侧成为y_{min}面，且v_{max}侧成为y_{max}面。NPC 的矩形实体部分（称为**三维视口**）在单位立方体内。

接着该单位立方体的$z = 1$面被映射到可被输出到显示器上的最大的正方形上。为了创建三维视口的内容的线框显示（它是视见体的内容），每一个输出图元的z分量被简单地抛弃，同时输出图元被显示。我们在第13章将会看到隐藏面的消除使用z分量来确定哪一个输出图元最接近于观察者并因此是可见的。

PHIGS使用两个4 × 4的矩阵，它们是视图方向矩阵和视图映射矩阵，来表示观察定义的完
202~205 整的集合。VRP、VPN和VUP被混合来形成**视图方向矩阵**，该矩阵将在空间坐标系中表示的位置转换为在VRC中表示的位置。这个变换使u、v和n轴分别对应x、y和z轴。

由PRP，u_{min}, u_{max}, v_{min}, v_{max}, F和B定义的视见体定义与由x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}定义的三维视口定义结合形成**视图映射矩阵**，该矩阵将VRC中的点转换为规格化投影坐标的点。形成视图方向矩阵和视图映射矩阵的子程序调用在7.3.4节讨论。

在6.4节，我们将看到怎样使用本节介绍的概念获得不同视图。6.5节将介绍平面几何投影的基础数学，而在6.6节将介绍为所有观察操作所需要的数学和算法。

6.4 三维观察的例子

本节我们考虑怎样应用前面介绍的基本观察概念来产生各种投影，如图6-17显示的那些投影。因为这些图中显示的房子在本节将被通篇使用，记住它的维度和位置（见图6-18）将是很有帮助的。对于每一个讨论的视图，我们给出一个显示VRP、VPN、VUP、PRP、窗口和投影类型（透视或平行）的表。本节假定NPC中的单位立方体为三维视口默

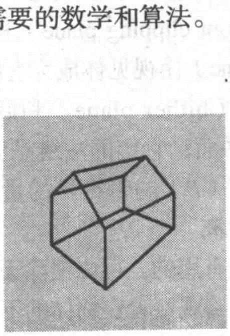


图6-17 房子的两点透视投影

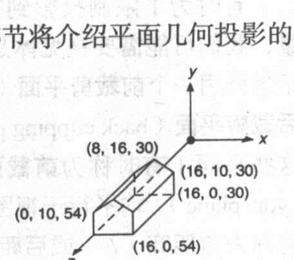


图6-18 这个房子在本章中作为世界坐标数据集的例子。这个房子在z轴上从30延伸到54，在x轴上从0延伸到16，在y轴上从0延伸到16

认值。符号(WC)或(VRC)添加到表中作为给定观察参数的坐标系的记号。为PHIGS使用的默认观察定义的表格的形式在此表出。默认值示于在图6-19a中。与这些默认值对应的视见体示于在图6-19b。如果投影类型是透视而不是平行，那么视见体是如图6-19c显示的四棱锥。 206

观察参数	值	注　释
VRP（WC）	(0，0，0)	原点
VPN（WC）	(0，0，1)	z 轴
VUP（WC）	(0，1，0)	y 轴
PRP（VRC）	(0.5,0.5,1.0)	
窗口（VRC）	(0,1,0,1)	
投影类型	平行	

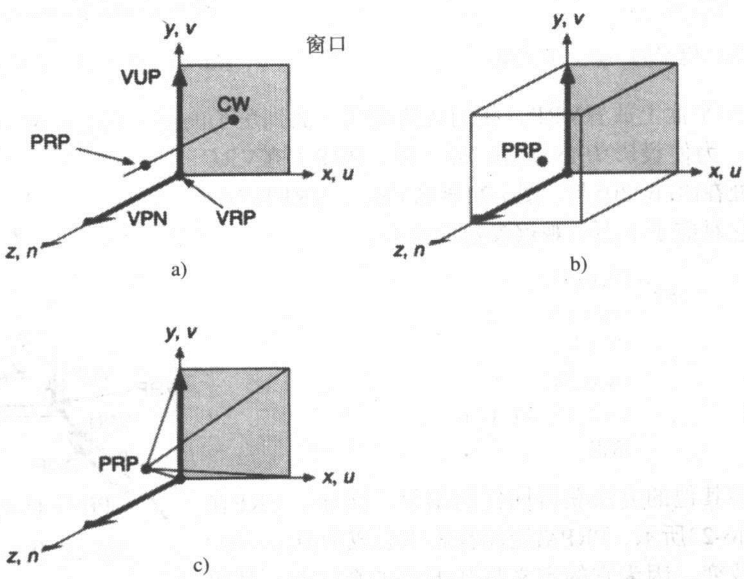


图6-19　观察参考坐标与世界坐标之间的关系。a) 默认的观察定义：VRP在原点，VUP是y轴，VPN是z轴。这使得u, v, n的VRC系统与x, y, z的世界坐标系统一致。窗口沿着u和v从0延伸到1，且PRP在(0.5, 0.5, 1.0)；b) 默认的平行投影视见体；c) 默认投影是透视投影的视见体

6.4.1　透视投影

为了得到如图6-20所示的房子（这个图和所有类似的图都用第7章讨论的SPHIGS程序绘制）前部的一点透视图，我们放置投影的中心（这可看成观察者的位置）到$x = 8$、$y = 6$和$z = 84$。x值被选择为房子的水平中心，y的值大约和一个站在（x, z）平面上的观察者的眼睛高度相等；z值任意。在这种情况下，将z值从房子的前面移动了30个单位（$z = 54$的平面）。窗口被设置得相当大，以保证房子适合在视见体内。所有其他的观察参数有其默认值，所有观察参数的整个集合如下： 207

VRP（WC）	(0, 0, 0)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(8, 6, 84)
窗口（VRC）	(−50, 50, −50, 50)
投影类型	透视

尽管图6-20的图像实际是房子的透视投影，但是它非常小且不在视图平面的中间。我们倾向

于房子的一个更居中的投影，同时使它的投影几乎覆盖整个视图平面，如图6-21所示。如果视图平面和房子的前平面重合，我们可以更轻松地得到这个效果。现在，由于房子的前平面在x和y两个方向上都从0延伸到16，如果窗口在x和y方向都从-1延伸到17，就可以得到合理的结果。

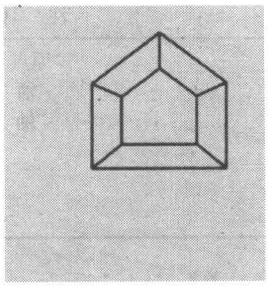


图6-20 房子的一点透视投影

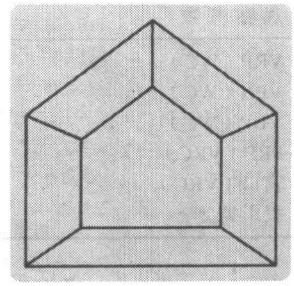


图6-21 房子的中心透视投影

通过在$z=54$平面上放置VRP，我们放置视图平面到房子的前平面上；(0, 0, 54)，即房子的前左下角即可。为使投影中心与图6-20一样，PRP（在VRC系统中）需要处在(8, 6, 30)上。图6-22显示VRC、VRP和PRP的新的安排，它对应于下面的观察参数的集合：

VRP（WC）	(0, 0, 54)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(8, 6, 30)
窗口（VRC）	(−1, 17, −1, 17)
投影类型	透视

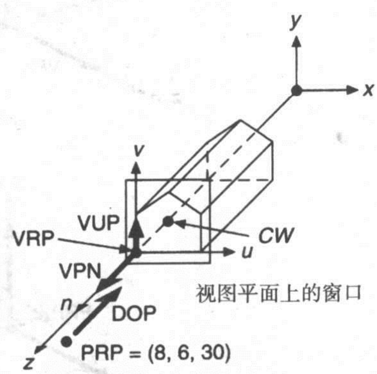


图6-22 图6-21的观察位置

可以用很多其他的方法获得同样的结果。例如，VRP在(8, 6, 54)，如图6-23所示，PRP给定的投影中心成为(0, 0, 30)。窗口必须也被改变，因为它的定义是基于VRC系统的，其原点为VRP。合适的窗口的u从-9延伸到9，v从-7延伸到11。对于房子，这是与前面例子使用相同的窗口，但是这里它定义于一个不同的VRC系统。因为向上的观察方向是y轴，所以u轴和x轴是平行的，就好像v轴和y轴是平行的一样。概括来讲，下面的观察参数，如图6-23所示，也产生图6-21：

VRP（WC）	(8, 6, 54)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(0, 0, 30)
窗口（VRC）	(−9, 9, −7, 11)
投影类型	透视

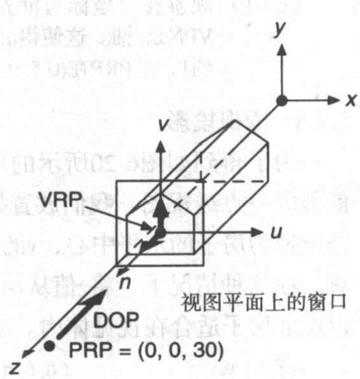


图6-23 图6-21的另一个观察位置

下面，让我们尝试获得如图6-17所示的两点透视投影。投影中心与取世界坐标物体快照的照相机的位置相似。考虑到这个相似性，图6-17的投影的中心好像位于房子的右上部，就好像从正z轴看一样。确切的投影中心是(36, 25, 74)。现在，如果房子在(16, 0, 54)的拐角选为VRP，则该投影中心是在相对于该角的(20, 25, 20)。由于视图平面与房子的前平面（$z=54$平面）重

合，一个u从 −20到20和v从 −5到35的窗口就肯定足够大到包含这个投影。因此，我们可以用如下的观察参数定义图6-24的视图：

VRP（WC）	(16, 0, 54)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(20, 25, 20)
窗口（VRC）	(−20, 20, −5, 35)
投影类型	透视

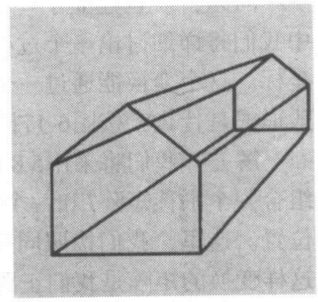


图6-24 从(36, 25, 74)看到的房子的透视投影，VPN平行于z轴

该视图类似于但是很明显不同于图6-17所示的视图。一方面，图6-17是一个两点透视投影，而图6-24是一个一点透视。很显然地，简单地移动投影中心不能产生图6-17。实际上，我们需要对视图平面重新定向，使得通过设置VPN为(1, 0, 1)，它可以交于x轴和z轴。因此，图6-17的观察参数如下：

VRP（WC）	(16, 0, 54)
VPN（WC）	(1, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	$(0, 25, 20\sqrt{2})$
窗口（VRC）	(−20, 20, −5, 35)
投影类型	透视

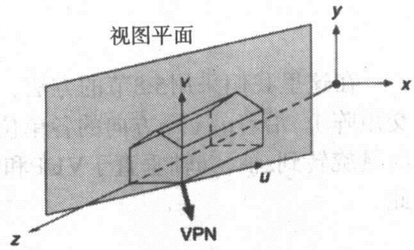


图6-25 对应于图6-17的视图平面和 VRC系统 208~209

图6-25显示带有该VPN建立的视图平面。使用了PRP的 $20\sqrt{2}$ n分量，以使投影中心距离在(x, y)平面的VRP的 $20\sqrt{2}$ 处，如图6-26所示。

有两种方式选择完全包围投影的窗口，像图6-17的窗口一样。可以用草图估计房子投影到视图平面的尺寸，如图6-26所示，来计算投影线与视图平面的交集。但是一个更好的方法是允许窗口绑定为程序中的变量，该程序被通过一个定值设备或定位设备交互确定。

图6-27从与图6-17相同的投影获得，但是窗口有一个不同的方向。在前面的所有例子中，VRC系统的v轴平行于世界坐标系中的y轴；因此窗口（其两侧面均平行于v轴）与房子的垂直面很好地一致。图6-27有与图6-17精确一致的观察参数，除了VUP从y轴旋转偏离了10°。

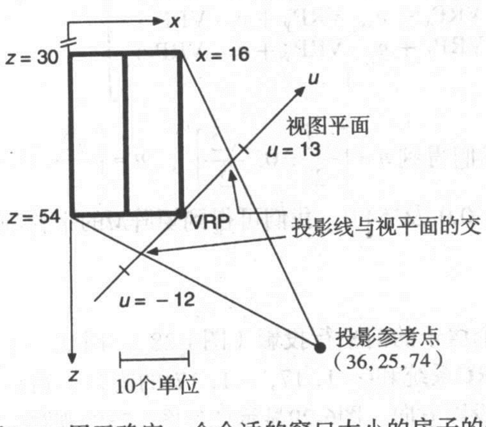


图6-26 用于确定一个合适的窗口大小的房子的俯视（平面）图

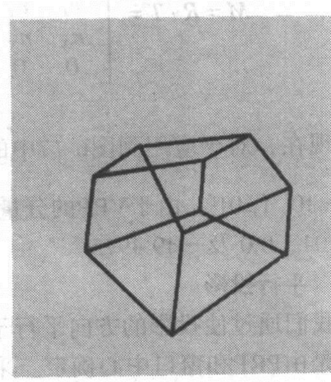


图6-27 由旋转VUP产生的房子的投影

例6.1

问题：一旦建立了一个VRC系统，所有后来的图形处理就在这个坐标系中进行，在6.6节中我们将详细讨论一个过程。先于那些处理步骤，必须将我们的数据集从世界坐标变换为VRC坐标。这个变换能通过一个单一矩阵进行，该矩阵能完成旋转和平移两种功能。这类矩阵的一般形式是什么？按图6-17描写的观察情况，矩阵的元素应有什么样的指定值？

解答：我们将采用5.8节及式(5-60)到式(5-64)提出的方法。那里，在x, y, z坐标系中，通过组合一个平移矩阵T和一个旋转矩阵R形成矩阵M，将任意一组直线平移和旋转到一个新的设定位置。这里，我们按照同样的过程，但要重新适应u, v, n坐标系以便和世界坐标系一致。实现
210 这样变换的矩阵是我们正要寻找的。

首先，我们必须平移VRC系统到原点。按照5.8节，我们用矩阵T实现该平移，T是简单的：

$$T=\begin{bmatrix}1&0&0&-\mathrm{VRP}_x\\0&1&0&-\mathrm{VRP}_y\\0&0&1&-\mathrm{VRP}_z\\0&0&0&1\end{bmatrix}$$

在这里我们采用5.8节的方法，那里我们找到一个由三个主轴确定的旋转矩阵（特殊的正交矩阵）。沿着u, v, n方向的各单位向量的分量，构成了这样一个矩阵的元素。因此，将VPN向量旋转到z轴，u轴垂直于VUP和VPN，v轴垂直于n和u，我们能找出旋转矩阵R的元素。因此，

$$n=\frac{\mathrm{VPN}}{\|\mathrm{VPN}\|},\quad u=\frac{\mathrm{VUP}\times\mathrm{VPN}}{\|\mathrm{VUP}\times\mathrm{VPN}\|},\quad v=n\times u$$

这里u, v, n表示单位向量。结果旋转矩阵是

$$R=\begin{bmatrix}u_x&u_y&u_z&0\\v_x&v_y&v_z&0\\n_x&n_y&n_z&0\\0&0&0&1\end{bmatrix}$$

因此我们寻找的单一矩阵是

$$M=R\cdot T=\begin{bmatrix}u_x&u_y&u_z&-(u_x\cdot\mathrm{VRP}_x+u_y\cdot\mathrm{VRP}_y+u_z\cdot\mathrm{VRP}_z)\\v_x&v_y&v_z&-(v_x\cdot\mathrm{VRP}_x+v_y\cdot\mathrm{VRP}_y+v_z\cdot\mathrm{VRP}_z)\\n_x&n_y&n_z&-(n_x\cdot\mathrm{VRP}_x+n_y\cdot\mathrm{VRP}_y+n_z\cdot\mathrm{VRP}_z)\\0&0&0&1\end{bmatrix}$$

现在，对于应用到图6-17中的指定值，我们得到$n=\left[\frac{\sqrt{2}}{2},0,\frac{\sqrt{2}}{2}\right]^{\mathrm{T}}$，$u=\left[\frac{\sqrt{2}}{2},0,-\frac{\sqrt{2}}{2}\right]^{\mathrm{T}}$及$v=[0,1,0]^{\mathrm{T}}$。由于VRP的分量分别是16.0，0.0 及 54.0，我们可得到矩阵M的平移项分别是26.8701，0.0及－49.4975。

6.4.2 平行投影

我们通过使投影的方向平行于z轴创建一个房子的前平行投影（图6-28）。回忆一下投影的方向是由PRP和窗口中心确定。采用默认的VRC系统和(－1, 17, －1, 17)的窗口，窗口中心是(8, 8, 0)。在(8, 8, 100)的PRP提供平行于z轴的投影方向。图6-29显示产生图6-28的观察位置。观
211 察参数如下：

VRP（WC）	（0,0,0）
VPN（WC）	（0,0,1）
VUP（WC）	（0,1,0）
PRP（VRC）	（8,8,100）
窗口（VRC）	（－1,17,－1,17）
投影类型	平行

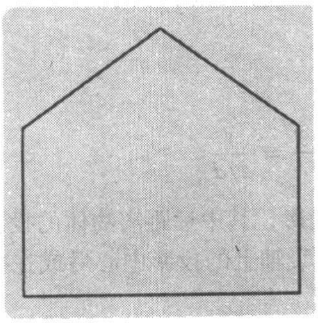


图6-28 房子的前平行投影

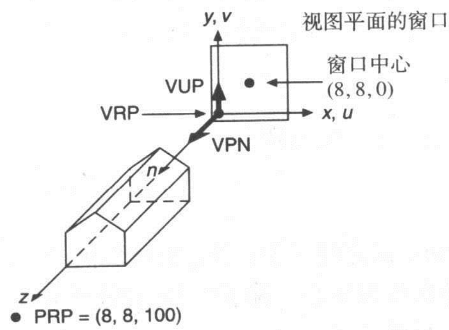


图6-29 产生图6-28所示房子的一个前视图的观察位置。PRP可以是$x = 8$和$y = 8$的任意点

为了创建侧视图，我们需要的观察位置，以(y, z)平面（或任何平行于它的平面）作为视图平面。我们通过使用(x, z)平面作为视图平面并且VPN作为y轴来创建一个房子的顶视图。但是，$+y$的默认的视图上方方向必须被改变；我们使用负x轴。

侧视图及顶视图的情况的完整的处理方法以及斜投影的例子请参见[FOLE90]。

6.4.3 有限的视见体

至此，在所有的例子中，视见体都被假设为无限的。6.3节描述的前和后裁剪平面可帮助确定一个**有限的视见体**。这些平面都是平行于视图平面的，且分别在距离视图参考点F和B的地方（沿着VPN从VRP开始度量）。为了避免负视见体，我们必须保证 F 在代数上大于B。

一个后墙裁剪掉的房子的前透视图（图6-30）是由下述观察定义（其中已经添加了F和B）导致的结果。如果给定距离，就假定是对相应平面做裁剪；否则，就不做。上述观察定义如下： 212

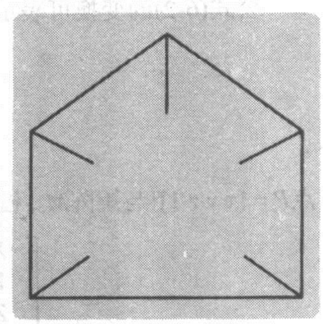


图6-30 后裁剪平面在$z = 31$处的房子的透视投影

VRP（WC）	（0,0,54）	房子的前左下角
VPN（WC）	（0,0,1）	z轴
VUP（WC）	（0,1,0）	y轴
PRP（VRC）	（8,6,30）	
窗口（VRC）	（－1,17,－1,17）	
投影类型	透视	
F (VRC)	+1	在房子前面一个单位，$z = 54 + 1 = 55$
B (VRC)	－23	在房子后面向前一个单位，$z = 54 - 23 = 31$

这种情况的观察位置与图6-22的一样，除了添加了裁剪平面。

如果前后裁剪平面动态移动，则被观察物体的三维结构可较静态观察时更容易识别。

6.5 平面几何投影的数学

在本节中，我们将介绍平面几何投影的基础数学。为了简化起见，我们假设用于透视投影

的投影平面与z轴垂直，且位于$z = d$处，而对于平行投影，投影平面是$z = 0$面。每一个投影可被4×4矩阵定义。这样很方便，因为投影矩阵可由变换矩阵组成，允许两个操作（变换，然后投影）被表示为一个矩阵。6.6节中我们将讨论任意投影平面。

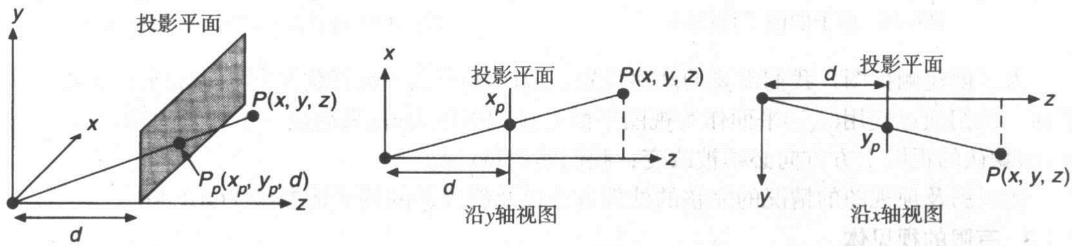
在本节中，我们从投影平面（该投影平面在位置$z = d$处平行于xy平面，在距离原点$|d|$处，有一个点P被投影到投影平面上）开始推导出几种投影的4×4矩阵。为了计算$P_p = (x_p, y_p, z_p)$，(x, y, z)在$z = d$处的投影平面上的透视投影，我们使用图6-31中的相似三角形来写比率

$$\frac{x_p}{d} = \frac{x}{z}; \quad \frac{y_p}{d} = \frac{y}{z} \tag{6-1}$$

用d乘以每一侧得到

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}, \quad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d} \tag{6-2}$$

距离d正好是应用于x_p和y_p的比例因子。被z除产生透视投影，其中远距离物体的投影比近处的物体的投影更小。除了0以外允许所有z值。投影点可在负z轴上的投影中心后或在投影中心和投影平面之间。



213 图6-31 透视投影

公式(6-2)的变换可表示为一个4×4的矩阵：

$$M_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{6-3}$$

点$P = [x\ y\ z\ 1]^{\mathrm{T}}$与矩阵$M_{\text{per}}$相乘产生一般齐次坐标的点$[X\ Y\ Z\ W]^{\mathrm{T}}$：

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{\text{per}} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6-4}$$

或

$$[X \quad Y \quad Z \quad W]^{\mathrm{T}} = \left[x \quad y \quad z \quad \frac{z}{d}\right]^{\mathrm{T}} \tag{6-5}$$

现在，被W（等于z/d）除，且去掉第四个坐标回到三维，我们有

$$\left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) = (x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d\right) \tag{6-6}$$

这些公式是式(6-1)的正确的结果，加上变换后的d的z坐标，d是沿着z轴的投影平面的位置。

另一个透视投影的公式将投影平面置于$z = 0$的位置，透视投影中心在$z = -d$，如图6-32所

示。三角形的相似性现在给出

$$\frac{x_p}{d} = \frac{x}{z+d}, \quad \frac{y_p}{d} = \frac{y}{z+d} \tag{6-7}$$

与d相乘，我们得到

$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}, \quad y_p = \frac{d \cdot y}{z+d} = \frac{y}{(z/d)+1} \tag{6-8}$$

投影矩阵是

$$M'_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \tag{6-9}$$

该公式允许d（到投影中心的距离）趋于无限。

在位于$z=0$的投影平面上的正投影是直接的。这种情况下，投影的方向与投影平面的法线（z轴）一致。因此，点P投影为

$$x_p = x, \quad y_p = y, \quad z_p = 0 \tag{6-10}$$

该投影用下面的矩阵表示：

$$M_{\text{ort}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-11}$$

注意当公式(6-9)中的d趋于无限时，公式(6-9)成为公式(6-11)。这是因为正投影是透视投影的特殊情况。

M_{per}仅应用在特殊的场合，即其投影中心在原点的情况。M_{ort}仅应用在投影方向平行于z轴的时候。[FOLE90]中引用的一个更一般的公式，不仅删除这些限制还将平行和透视投影统一到一个公式中。

本节中，我们已经看到怎样表示M_{per}，M'_{per}和 M_{ort}，所有这些都是投影平面垂直于z轴的情况。在6.6节中，我们去除这个限制且考虑有限视见体隐含的裁剪。

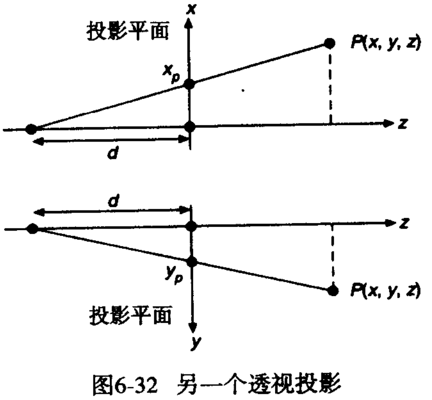


图6-32 另一个透视投影

例6.2

问题：矩阵M_{per}定义了一点透视投影。试描述定义两点透视投影的矩阵，以及它与我们推导的矩阵M_{per}的关系。定义三点透视投影的矩阵的形式是什么？

解答：如6.4.1节中所建议那样，我们需要定向视图平面，使得它能切割多于一个轴（这里是z轴）。例如，我们将指定一个绕y轴的旋转，使得视图平面与x和z两个轴相交。通过将下列矩阵后乘矩阵M_{per} 我们可得到新矩阵： 214~215

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

这里θ是绕y轴的旋转角度。结果矩阵为

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ -\sin\theta/d & 0 & \cos\theta/d & 0 \end{bmatrix}$$

注意合成矩阵的a_{41}位置有非零项出现。它表明有一个灭点在x轴上。如果我们绕x轴实现一个相似的旋转，则a_{42}位置的非零项表明有一个灭点在y轴上。对于x及y轴的旋转组合可产生一个三点透视。

6.6 实现平面几何投影

给定一个视见体和一个投影，让我们考虑裁剪和投影的观察操作怎样被实际应用。正如观察过程的概念模型（图6-2）所提示的那样,我们可以通过计算定义视见体的六个面的每一个与直线的交点来裁剪直线。在裁剪后仍存在的线将被投影到视图平面上，这一过程要同时求解投影线与视图平面的交点方程。然后交点坐标从三维世界坐标转化到二维设备坐标。但是，该过程需要的大量计算过程（很多线需要重复）包括相当大的计算量。令人高兴的是，有一个更有效的过程，基于分治策略将一个困难的问题分解为一系列简单的问题。

某些视见体较一般的空间更容易被裁剪（裁剪算法在6.6.3节被讨论）。例如，计算直线和被六个面

$$x=-1, \quad x=1, \quad y=-1, \quad y=1, \quad z=0, \quad z=-1 \tag{6-12}$$

定义的平行投影视见体的平面的交点是很简单的。这对于下述平面定义的透视投影视见体也成立：

$$x=z, \quad x=-z, \quad y=z, \quad y=-z, \quad z=-z_{\min}, \quad z=-1 \tag{6-13}$$

216 这些**规范视见体**示于图6-33。

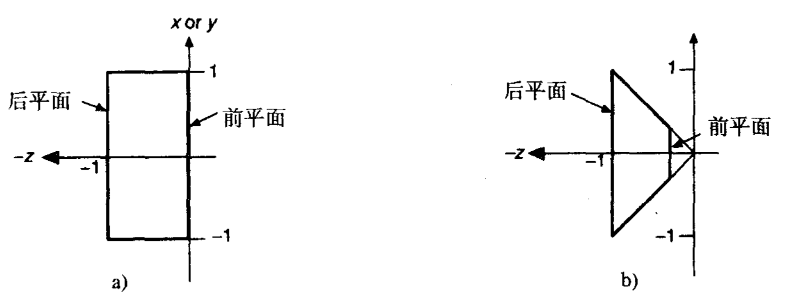


图6-33 两个规范视见体。a) 为平行投影，b) 为透视投影

我们的策略是找到**规格化变换**N_{par}和N_{per}，它们将任意的平行投影或透视投影视见体分别转换为平行和透视规范视见体。然后执行裁剪，接着通过6.5节的矩阵投影到二维。该策略的风险是需要浪费很多随后要被裁剪操作抛弃的点也要做变换，但是至少裁剪变得容易做了。

图6-34显示这里包括的一系列过程。我们可以通过组合步骤3和步骤4为一个变换矩阵将它简化为变换-裁剪-变换序列。对透视投影，还需要一个除法来从齐次坐标映射回三维坐标。该除法在组合序列的第二个变换后完成。另一个可选择的策略，在齐次坐标中裁剪，将在6.6.4节中讨论。

熟悉PHIGS的读者会注意到式(6-12)和式(6-13)的规范视见体与PHIGS的**默认视见体**不同：对平行投影，是x, y, z从0到1的单位立方体，而对于透视投影，是顶点在(0.5, 0.5, 1.0)且侧面穿

过在$z=0$的平面上x和y从0到1的单位正方形的四棱锥。这里的规范视见体的定义是为简化裁剪公式且提供6.6.4节讨论的平行投影和透视投影之间的一致性。另一方面，PHIGS默认视见体的定义使二维观察成为三维观察的一种特殊情况。

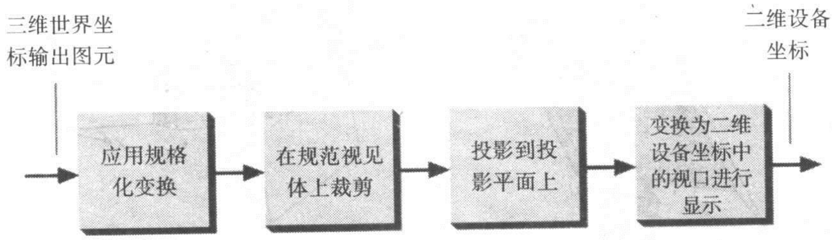


图6-34 三维观察的实现

在6.6.1节和6.6.2节中，我们为透视投影和平行投影推导规格化变换，它用做变换-裁剪-变换序列的第一步。

6.6.1 平行投影

本节中，我们为平行投影推导规格化变换N_{par}来变换世界坐标位置，使得视见体变换为由式(6-12)定义的规范视见体。变换后的坐标按照该规范视见体裁剪，其结果投影到$z=0$平面，然后变换到视口来显示。 217

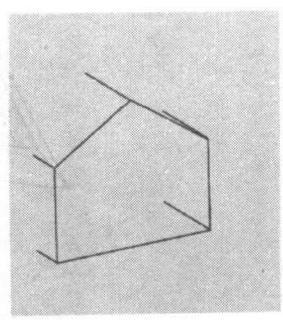


图6-35 裁剪后的房子的最终平行投影

变换N_{par}按最一般的情况（即斜（而不是正）平行投影）推导。因此N_{par}包括一个导致观察坐标的投影方向平行于z的错切转换，即使在（u, v, n）坐标系它也并不平行于VPN。通过包括该错切，我们可以仅通过设置$z=0$来实现在$z=0$面上的投影。如果平行投影是正平行投影，则该规格化变换的错切分量是恒等的。

组成N_{par}的一系列变换如下：

1) 平移VRP 到原点。

2) 旋转VRC使n轴（VPN）成为z轴，u轴成为x轴，v轴成为y轴。

3) 错切变换使投影方向平行于z轴。

4) 平移和缩放变换到由公式(6-12)给出的平行投影规范视见体。

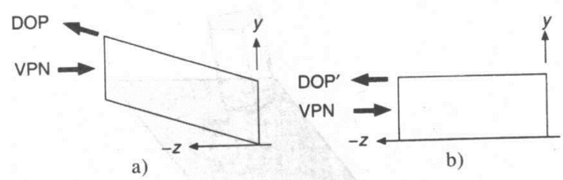


图6-36 使用视见体的侧视图作为错切的例子。a) 中的平行四边形错切到 b) 中的矩形；由于VPN平行于z轴，所以不变

在PHIGS中，步骤1和步骤2定义**视图方向矩阵**（view-orientation matrix），同时步骤3和步骤4定义**视图映射矩阵**（view-mapping matrix）。

图6-37显示该变换序列应用到平行投影视见体和一个房子的轮廓的情形；图6-35显示上述变换序列产生的平行投影。

步骤1正好是平移变换$T(-\mathrm{VRP})$。对于步骤2，我们使用5.3节和5.7节讨论过的并在式(5-64)和式(5-65)推导中表明的正交矩阵的性质。执行步骤2的旋转矩阵的行向量是单位向量，它们被R旋转到x轴，y轴和z轴。VPN旋转到z轴，于是

$$R_z = \frac{\mathrm{VPN}}{\|\mathrm{VPN}\|} \tag{6-14}$$

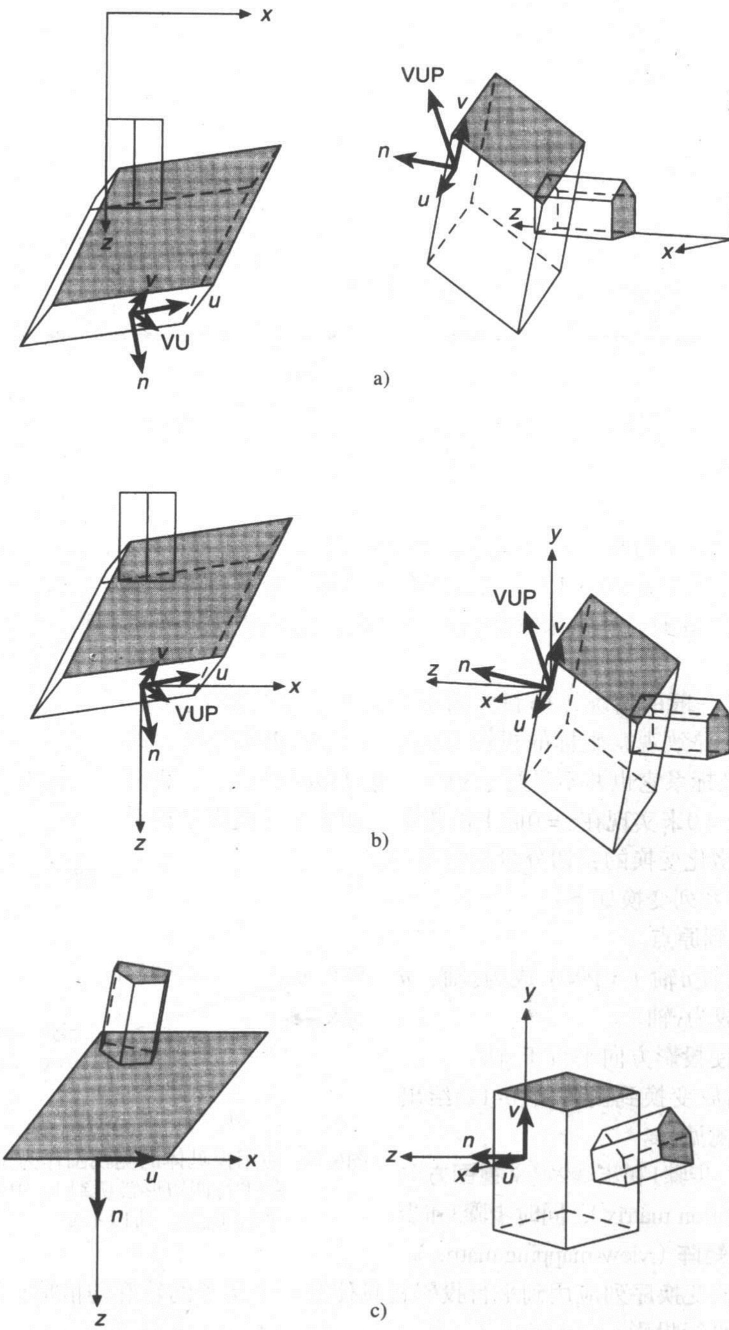
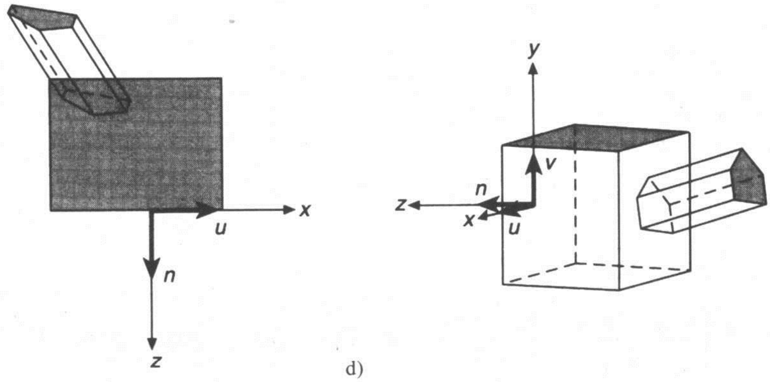
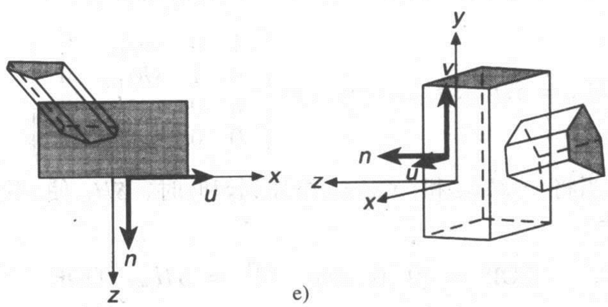


图6-37 在平行投影观察流水线的各个阶段的结果。在每种情况都显示一个俯视的偏轴平行投影。a) 最初的观察位置，b) VRP平移到原点，c) (u, v, n) 坐标系旋转到与（x, y, z）系统一致，d) 视见体错切使得投影方向(DOP)与z轴平行，e) 视见体平移和缩放变换到规范平行投影视见体。观察参数是VRP = (0.325, 0.8, 4.15)，VPN = (0.227, 0.267, 1.0)，VUP = (0.293, 1.0, 0.227)，PRP = (0.6, 0.0, – 1.0)，窗口 = (– 1.425, 1.0, – 1.0, 1.0)，F = 0.0，B = – 1.75。（图由乔治·华盛顿大学的L.Lu编程绘出。）



d)



e)

图6-37 （续）

u轴（垂直于VUP和VPN, 因此是单位向量沿着VUP 和R_z（与VPN方向相同）的叉积。）被旋转到x轴，于是

$$R_x = \frac{\text{VUP} \times R_z}{\| \text{VUP} \times R_z \|} \tag{6-15}$$

类似地，v轴（垂直于R_z和R_x）,旋转到y轴，于是

$$R_y = R_z \times R_x \tag{6-16}$$

因此，步骤2的旋转由下述矩阵给出：

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-17}$$

其中r_{1x}是R_x的第一个元素，依此类推。

第三步是沿着z轴错切视见体，使得它的所有平面垂直于坐标轴之一。我们通过确定错切应用到投影方向（DOP）使其与z轴重合来实现该步骤。回忆一下DOP是从PRP 到窗口中心（CW）的向量，并且PRP定义于VRC系统。最前面的两个变换步骤使VRC与世界坐标系一致，所以PRP自身现在是在世界坐标中。因此，DOP是CW－PRP。给定

$$\text{DOP} = \begin{bmatrix} dop_x \\ dop_y \\ dop_z \\ 0 \end{bmatrix}, \quad \text{CW} = \begin{bmatrix} \dfrac{u_{\max} + u_{\min}}{2} \\ \dfrac{v_{\max} + v_{\min}}{2} \\ 0 \\ 1 \end{bmatrix}, \quad \text{PRP} = \begin{bmatrix} prp_u \\ prp_v \\ prp_n \\ 1 \end{bmatrix} \tag{6-18}$$

于是

$$\begin{aligned} \text{DOP} &= \text{CW} - \text{PRP} \\ &= \left[\frac{u_{\max} + u_{\min}}{2} \quad \frac{v_{\max} + v_{\min}}{2} \quad 0 \quad 1\right]^{\text{T}} - [prp_u \quad prp_v \quad prp_n \quad 1]^{\text{T}} \end{aligned} \tag{6-19}$$

图6-36显示这样定义的DOP和预期的DOP'。

错切可用5.7节的式(5-45)的(x, y)错切矩阵完成。使用系数shx_{par}和shy_{par}，矩阵是

$$SH_{\text{par}} = SH_{xy}(shx_{\text{par}}, shy_{\text{par}}) = \begin{bmatrix} 1 & 0 & shx_{\text{par}} & 0 \\ 0 & 1 & shy_{\text{par}} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-20}$$

如5.7节描述的那样，当将$z \cdot shx_{\text{par}}$和$z \cdot shy_{\text{par}}$添加到x和y时，SH_{xy}使z不受影响。我们要求出shx_{par}和shy_{par}使得

$$\text{DOP}' = [0 \quad 0 \quad dop_z \quad 0]^{\text{T}} = SH_{par} \cdot \text{DOP} \tag{6-21}$$

218~221 执行式(6-21)的乘法，随后进行代数处理，表明等式发生在

$$shx_{\text{par}} = -\frac{dop_x}{dop_z}, \qquad shy_{\text{par}} = -\frac{dop_y}{dop_z}. \tag{6-22}$$

注意到，对于正投影，$dop_x = dop_y = 0$,于是$shx_{\text{par}} = shy_{\text{par}} = 0$,且错切矩阵简化为恒等矩阵。

图6-38显示这三个变换步骤应用后的视见体。视见体的范围是

$$u_{\min} \leqslant x \leqslant u_{\max}, \quad v_{\min} \leqslant y \leqslant v_{\max}, \quad B \leqslant z \leqslant F \tag{6-23}$$

这里F和B分别是从VRP沿着VPN到前裁剪平面和后裁剪平面的距离。

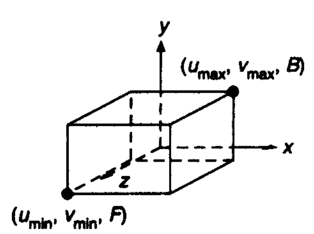


图6-38 经过变换步骤1到步骤3后的视见体

在过程中的第四步即最后一步是把错切变换后的视见体变换到规范视见体。我们通过平移式(6-23)给出的视见体的前中心到原点，然后缩放变换到式(6-12)给出的$2 \times 2 \times 1$的最终规范视见体来实现该步骤。这两个变换是

$$T_{\text{par}} = T\left(-\frac{u_{\max} + u_{\min}}{2}, -\frac{v_{\max} + v_{\min}}{2}, -F\right) \tag{6-24}$$

$$S_{\text{par}} = S\left(\frac{2}{u_{\max} + u_{\min}}, \frac{2}{v_{\max} + v_{\min}}, \frac{1}{F - B}\right) \tag{6-25}$$

如果F和B没有定义（因为前平面裁剪和后平面裁剪不起作用），那么任意满足$B \leqslant F$的值可被使用。值0和1是满足的。

总体来说，我们有

$$N_{\text{par}} = S_{\text{par}} \cdot T_{\text{par}} \cdot SH_{\text{par}} \cdot R \cdot T(-\text{VRP}) \tag{6-26}$$

N_{par}把任意的平行投影视见体变换到平行投影规范视见体，因此允许输出图元在平行投影规范视见体上裁剪。

6.6.2 透视投影

我们现在为透视投影推导规格化变换N_{per}。N_{per}变换世界坐标位置，使得视见体成为透视规范视见体，即截断的四棱锥，其顶点是由式(6-13)定义的原点。在应用N_{per}后，裁剪在规范视见体进行，裁剪结果用（6.5节推导的）M_{per}投影到视图平面。

组成N_{per}的变换序列如下：

1) 平移VRP到原点。

2) 旋转VRC 使得n轴（VPN）成为z轴，u轴成为x轴，v轴成为y轴。

3) 平移使得PRP给定的投影中心（COP）在原点。 222

4) 错切使得视见体的中心线成为z轴。

5) 缩放变换使得视见体成为规范透视视见体，即由公式(6-13)的六个平面定义的截断四棱锥。

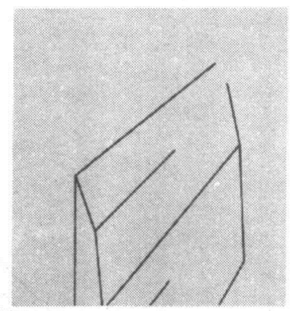


图6-39 被裁剪的房子的最终透视投影

图6-41显示这个变换序列应用于透视投影视见体和一个房子。图6-39显示结果透视投影。

步骤1和步骤2 是与平行投影一致的：$R \cdot T(-\text{VRP})$。步骤3是投影中心(COP)到原点的平移，为规范视见体所要求。COP由PRP = (prp_u, prp_v, prp_n)在VRC中相对于VRP定义。观察参考坐标（VRC）由步骤1和步骤2变换到世界坐标，于是在VRC中COP的定义现在也在世界坐标中。因此，步骤3的平移正好是$T(-\text{PRP})$。

为了计算第4步的错切，我们研究图6-40，该图显示在步骤1到步骤3的变换后视见体的侧视图。注意，视见体的中心线（它穿过原点和窗口中心）与$-z$轴不同。错切的目的是变换中心线为$-z$轴。视见体的中心线从PRP(它现在是在原点)到CW（窗口中心）。因此它与平行投影的投影方向相同，即CW − PRP。因此错切矩阵是SH_{par}，与平行投影相同！考虑这个问题的另一种方式是在步骤3中平移−PRP，即将投影中心放在原点，也将CW平移−PRP，于是步骤3后，视见体的中心线穿过原点和CW − PRP。

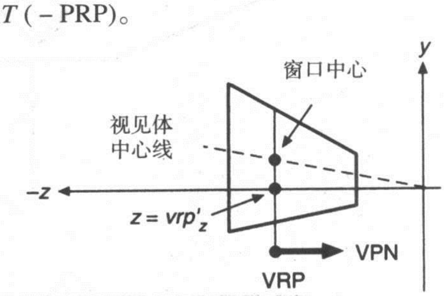


图6-40 变换步骤1到步骤3后的视见体的横截面

在应用错切后，窗口（因此是视见体）集中在z轴上。在投影平面上窗口的范围是

$$-\frac{u_{\max} - u_{\min}}{2} \leqslant x \leqslant \frac{u_{\max} - u_{\min}}{2} \tag{6-27}$$

$$-\frac{v_{\max} - v_{\min}}{2} \leqslant y \leqslant \frac{v_{\max} - v_{\min}}{2}$$ 223

VRP在步骤3前是在原点，现在由步骤3平移且由步骤4错切。定义VRP'作为在步骤3和步骤4的变换后的VRP，

$$\text{VRP}' = SH_{\text{par}} \cdot T(-\text{PRP}) \cdot [0 \;\; 0 \;\; 0 \;\; 1]^{\mathrm{T}} \tag{6-28}$$

VRP'的z分量（表示成vrp_z'）与$-prp_n$相等，这是因为(x, y)平面的错切SH_{par}并不影响z坐标。

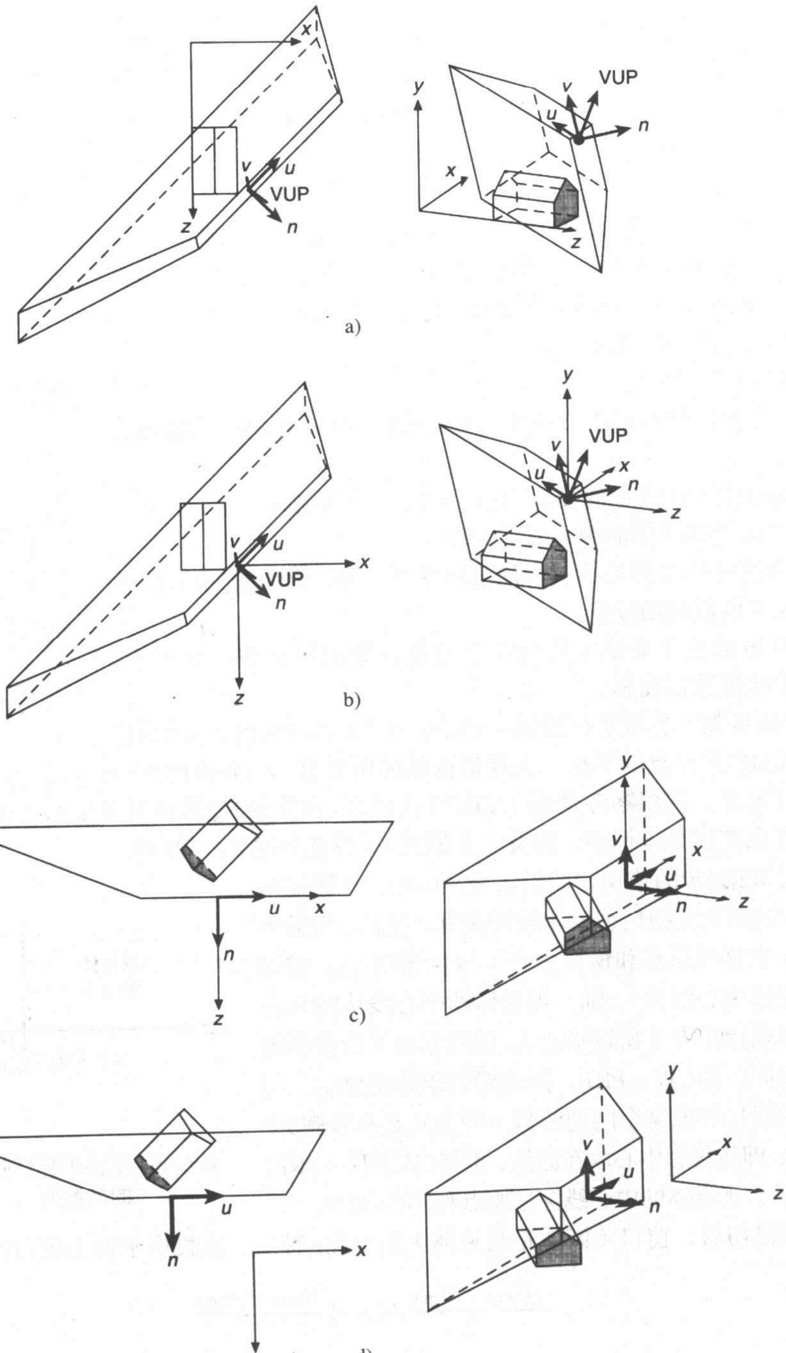


图6-41 透视投影观察流水线的各个阶段的结果。在每种情况下都显示一个俯视偏轴平行投影。a)最初的观察位置，b)VRP平移到原点，c)(u, v, n) 坐标系统旋转到与(x, y, z)坐标系统一致。d)投影中心（COP）平移到原点，e)视见体错切使得投影方向(DOP)与z轴平行，f)缩放变换视见体到规范透视投影视见体。观察参数是VRP = (1.0, 1.275, 2.6)，VPN = (1.0, 0.253, 1.0)，VUP = (0.414, 1.0, 0.253)，PRP = (1.6, 0.0, 1.075)，窗口 = (– 1.325, 2.25, – 0.575, 0.575)，$F = 0$，B = – 1.2。（此图由乔治·华盛顿大学的L.Lu编程绘出。）

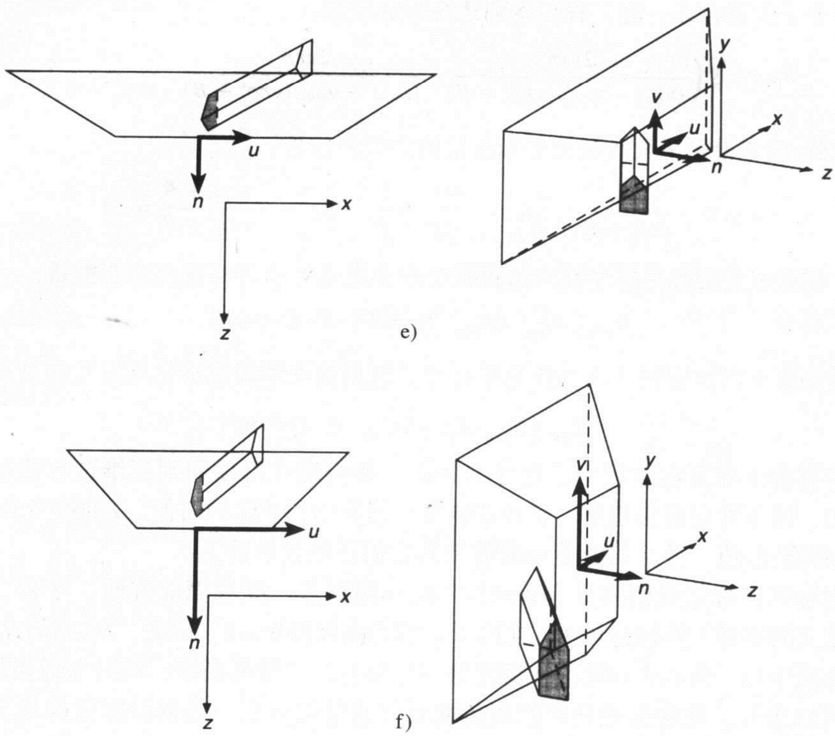


图6-41 （续）

最后一个步骤是沿着三个轴缩放变换以产生由式(6-13)定义并示于图6-42的规范视见体。因此，缩放变换最好想像成是在两个子步骤里实现。在第一个子步骤里，我们按不同比例变换x和y，以给出界定视见体单位斜度边界的倾斜面。我们通过缩放窗口使得它的半高度和半宽度都是$-vrp_z'$来完成该子步骤。合适的x和y缩放因子分别是$-2 \cdot vrp_z'/(u_{max}-u_{min})$和$-2 \cdot vrp_z'/(v_{max}-v_{min})$。在第二个子步骤里，我们沿着三个轴（以保持单位斜率）均匀缩放使得在$z=vrp_z'+B$的后裁剪平面成为$z=-1$平面。该子步骤的缩放因子是$-1/(vrp_z'+B)$。该缩放因子具有一个负号，因为$vrp_z'+B$本身为负，使缩放因子成为正的。

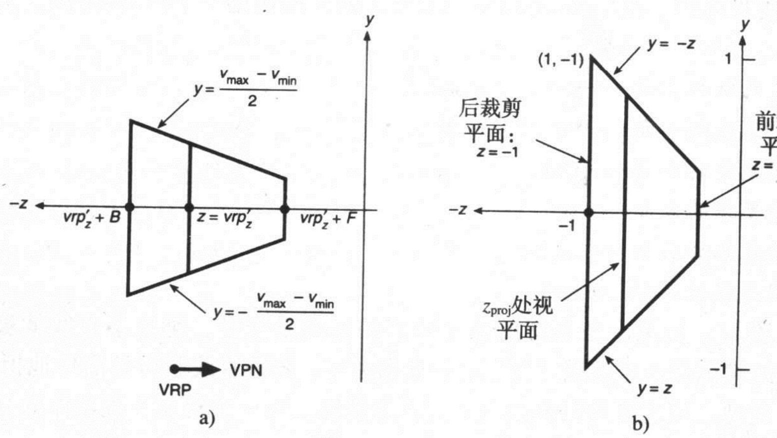


图6-42 在最后的缩放步骤前后的视见体的横截面。在这个例子中，F和B的符号相反，所以前裁剪平面和后裁剪平面在VRP的相反侧面。a) 缩放前，b) 缩放后

将这两个子步骤合在一起，我们得到透视缩放变换：

224~226

$$S_{per}= S\left(\frac{2vrp'_z}{(u_{max}-u_{min})(vrp'_z+B)},\frac{2vrp'_z}{(v_{max}-v_{min})(vrp'_z+B)},\frac{-1}{vrp'_z+B}\right) \tag{6-29}$$

应用该缩放到z改变投影平面和裁剪平面的位置到新的位置：⊖

$$z_{proj}=-\frac{vrp'_z}{vrp'_z+B},\quad z_{min}=-\frac{vrp'_z+F}{vrp'_z+B},\quad z_{max}=-\frac{vrp'_z+B}{vrp'_z+B}=-1 \tag{6-30}$$

总体来说，将透视投影视见体变换为透视投影规范视见体的规格化观察变换是：

$$N_{per}=S_{per}\cdot SH_{par}\cdot T(-\mathrm{PRP})\cdot R\cdot T(-\mathrm{VRP}) \tag{6-31}$$

类似地，回忆将平行投影视见体变换为平行投影规范视见体的规格化观察变换是：

$$N_{par}=S_{par}\cdot T_{par}\cdot SH_{par}\cdot R\cdot T(-\mathrm{VRP}) \tag{6-26}$$

这些变换发生在齐次空间中。在什么条件下我们现在可以回到三维进行裁剪呢？只要我们知道$W>0$。该条件很容易理解。负W意味着：当我们用W除的时候，Z的符号和z将相反。具有负Z的点将有正z值，并且可能显示时看上去它们已经裁剪掉了。

什么时候可以肯定我们将具有$W>0$呢？被应用到点、线和面的旋转、平移、缩放和错切（如第5章定义的那样）将保持$W>0$；实际上，它们将保持$W=1$。因此，N_{per}或者N_{par}都不会影响变换点的齐次坐标，所以被W除正常情况将不是映射回三维所必需，同时在适当的规范视见体上的裁剪可以进行。在根据透视投影规范视见体进行裁剪后，透视投影矩阵M_{per}（包含除法）必须应用。

如果输出图元包括表示成齐次坐标函数的曲线和曲面，并显示为连接起来的直线段，则可能得到$W<0$。例如，如果X的符号不改变，W的函数的符号从曲线的一个点到下一个点发生变化，那么X/W将在曲线两个点上有不同的符号。在第9章讨论的有理B样条是其中一例。$-W$也产生于使用一些在第5章讨论的变换之外的变换，例如伪影（“fake” shadow）[BLIN88]。在6.6.3节，将讨论几个三维空间裁剪算法。然后，在6.6.4节我们讨论当我们不能确保$W>0$时怎样裁剪。

6.6.3 用三维规范视见体进行裁剪

对于平行投影来说，规范视见体是$2\times2\times1$棱柱，对于透视投影来说则是截断的规则四棱

227 锥。在第3章讨论的Cohen-Sutherland和Cyrus-Beck裁剪算法很容易扩展到三维。

对于规范平行视见体的二维Cohen-Sutherland算法的扩展使用一个6位的外码。当满足条件时，某一位为真(1)：

第1位—点在视见体的上面——$y>1$

第2位—点在视见体的下面——$y<-1$

第3位—点在视见体的右面——$x>1$

第4位—点在视见体的左面——$x<-1$

第5位—点在视见体的后面——$z<-1$

第6位—点在视见体的前面——$z>0$

与二维的情况类似，如果一条直线的两个端点的编码是全零，则这条直线就被简单地接受；如果两个端点的编码逐位求逻辑“与”后并非全为零，则该直线就可被简单地拒绝。否则，将开始进一步分割处理。可能要进行多达六个交点的计算，每一个对应于视见体的一个侧面。

⊖ z_{min}和z_{max}是根据它们的绝对值关系命名的，z_{min}代数上比z_{max}大。

求交计算使用从$P_0(x_0, y_0, z_0)$到$P_1(x_1, y_1, z_1)$的线段的参数表示：

$$x = x_0 + t(x_1 - x_0) \tag{6-32}$$

$$y = y_0 + t(y_1 - y_0) \tag{6-33}$$

$$z = z_0 + t(z_1 - z_0) \qquad 0 \leqslant t \leqslant 1 \tag{6-34}$$

当t可以从0变到1时，这三个方程给出了直线段上从P_0到P_1所有点的坐标。

为了计算一条直线与视见体的$y = 1$平面的交点，我们用常数1替代式(6-33)中的变量y, 从而求出t, $t = (1 - y_0)/(y_1 - y_0)$。如果$t$在0到1这一区间的外面，则交点在从点$P_0$到$P_1$的直线的无限远处，但是不在$P_0$和$P_1$之间的直线段上，因此不是所求的。如果$t$在[0, 1]区间，则将$t$代入求$x$和$z$，从而得到交点的坐标：

$$x = x_0 + \frac{(1 - y_0)(x_1 - x_0)}{y_1 - y_0}, \qquad z = z_0 + \frac{(1 - y_0)(z_1 - z_0)}{y_1 - y_0} \tag{6-35}$$

该算法使用外码使得t是否在[0,1]内的检测成为不必要的。

用于规范化透视视见体裁剪的外码位如下：

第1位—点在视见体的上面——$y > -z$

第2位—点在视见体的下面——$y < z$

第3位—点在视见体的右面——$x > -z$

第4位—点在视见体的左面——$x < z$

第5位—点在视见体的后面——$z < -1$

第6位—点在视见体的前面——$z > z_{min}$

计算直线和斜平面的交点是简单的。在$y = z$面上，式(6-33)必须与式(6-34)相等，$y_0 + t\ (y_1 -$
$y_0) = z_0 + t\ (z_1 - z_0)$。于是 228

$$t = \frac{z_0 - y_0}{(y_1 - y_0) - (z_1 - z_0)} \tag{6-36}$$

将t代入公式(6-32）和公式(6-33)，则得到

$$x = x_0 + \frac{(x_1 - x_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}, \qquad y = y_0 + \frac{(y_1 - y_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)} \tag{6-37}$$

我们知道$z = y$。选择该规范视见体的原因现在很清楚：平面的单位斜率使得交点的计算比任意斜率平面时的计算简单。

有另外几种基于直线的参数表达式的裁剪算法[CYRU78; LIAN84]，并且这些算法比简单的Cohen-Sutherland算法更有效。参见[FOLE90]的第6章和第19章。

6.6.4 在齐次坐标中裁剪

在齐次坐标中进行裁剪有两个原因。第一个与效率有关：有可能将透视投影规范视见体变换为平行投影规范视见体，所以总可使用为平行投影规范视见体而优化的单裁剪过程。但是，这个裁剪必须在齐次坐标中完成以保证正确的结果。在观察操作的硬件实现中可提供这种单裁剪过程。第二个原因是作为非寻常齐次变换的结果出现的和来自使用有理参数样条（第9章）的点可能具有负的W，这些点可以在齐次坐标而不是三维中被正确裁剪。

对于裁剪，可以证明从透视投影规范视见体到平行投影规范视见体的变换是

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad 0 > z_{\min} > -1 \tag{6-38}$$

回想式(6-30) $z_{\min} = -(vrp_z' + F)/(vrp_z' + B)$和式(6-28)$VRP' = SH_{par} \cdot T(-PRP) \cdot [0\ 0\ 0\ 1]^T$。图6-43显示将$M$应用到透视投影规范视见体的结果。

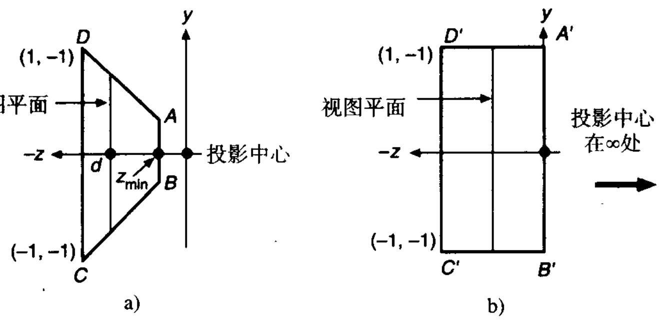


图6-43 在应用矩阵M前a)和后b)的规格化透视视见体的侧视图

矩阵M与透视投影规格化变换N_{per}集成：

$$N'_{per} = M \cdot N_{per} = M \cdot S_{per} \cdot SH_{par} \cdot T(-PRP) \cdot R \cdot T(-VRP) \tag{6-39}$$

通过对透视投影使用N'_{per}代替N_{per}，同时通过对平行投影连续使用N_{par}，我们可以在平行投影规范视见体中进行裁剪而不是在透视投影规范视见体中进行裁剪。

三维平行投影视见体由$-1 \leqslant x \leqslant 1, -1 \leqslant y \leqslant 1, -1 \leqslant z \leqslant 0$定义。通过用$X/W$代替$x$，$Y/W$代替$y$，$Z/W$代替$z$，我们求出齐次坐标中的对应的不等式，它可导致

$$-1 \leqslant X/W \leqslant 1, \quad -1 \leqslant Y/W \leqslant 1, \quad -1 \leqslant Z/W \leqslant 0 \tag{6-40}$$

对应的平面方程是

$$X = -W, \quad X = W, \quad Y = -W, \quad Y = W, \quad Z = -W, \quad Z = 0 \tag{6-41}$$

为了理解怎样使用这些约束和平面，我们必须分别考虑$W>0$和$W<0$的情况。在第一种情况中，我们可以将W与式(6-40)的不等式相乘而不改变不等式的方向。在第二种情况下，相乘改变了不等式的方向。结果可表示为

$$W > 0: \quad -W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W, \quad -W \leqslant Z \leqslant 0 \tag{6-42}$$

$$W < 0: \quad -W \geqslant X \geqslant W, \quad -W \geqslant Y \geqslant W, \quad -W \geqslant Z \geqslant 0 \tag{6-43}$$

在下面的情况中，即裁剪一般的线和点，仅需要使用式(6-42)给定的区域，这是因为在使用M以前，所有的可见点都有$W>0$（通常$W=1$）。

但是，正如我们将在第9章看到的那样，有时需要直接在齐次坐标中用任意的W坐标表示点。因此，我们可能有$W<0$，意味着裁剪必须在式(6-42)和式(6-43)给出的区域内实现。图6-44将这些区域显示为A和B，同时也表明为什么必须使用两个区域。

在区域A中的点$P_1 = [1\ 3\ 2\ 4]^T$变换到三维点(1/4, 3/4, 2/4)，该点在规范视见体$-1 \leqslant x \leqslant 1$,
229~230 $-1 \leqslant y \leqslant 1$，$-1 \leqslant z \leqslant 0$中。点$P_2 = -P_1 = [-1\ -3\ -2\ -4]^T$（不在区域$A$内，但在区域$B$内），变换到与$P_1$同样的三维点，即（1/4，3/4，2/4)。如果仅对区域A裁剪，则P_2将被不正确地抛弃。

这个可能性是存在的，由于齐次坐标点P_1和P_2相差一个常数乘子(－1)，同时我们知道这些齐次点对应于同样的三维点（在齐次空间的$W = 1$平面上）。

对于区域B上的点，这个问题有两种解决方法。一种是裁剪所有的点两次，一次裁剪一个区域。但是做两次裁剪是很昂贵的。更好的一个解决方法是先用$-W$使点反号，如对于P_2，然后裁剪它们。同样地，我们可以通过对每个端点乘以-1来正确地裁剪一条端点都在图6-44的区域B内的线，将点放在区域A内。

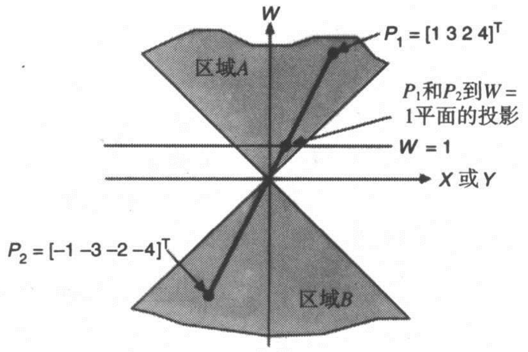


图6-44 点P_1和P_2都映射到$W = 1$平面上的同一个点，在穿过原点和上述两点的直线上所有的点都类似。在齐次坐标中按照区域A的裁剪将不正确地抛弃P_2

对于直线，如P_1P_2（如图6-45所示，其端点有相反符号的W值），另一个问题产生了。直线到$W = 1$平面的投影是两条线段，一条趋于正无穷，另一条趋于负无穷。现在的解决方法是裁剪两次，一次对一个区域进行裁剪。有可能每一次裁剪将返回一个可见线段。一个简单的方法是在区域A内裁剪该直线，再使直线的两个端点反号，再次对区域A裁剪。该方法保留了原来在齐次坐标中裁剪的目的之一：使用一个简单的裁剪区域。有兴趣的读者可参考[BLIN78a]做进一步的讨论。

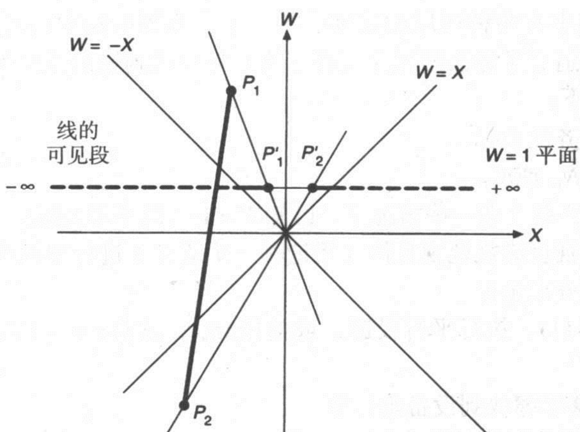


图6-45 线段P_1P_2投影到两个直线段，一个从P_2'到正无穷，另一个从P_1'到负无穷（实粗线表示在裁剪区域内，虚粗线表示在裁剪区域外）。该线段必须裁剪两次，对一个区域裁剪一次

给定式(6-41)，Cohen-Sutherland 或Cyrus-Beck算法可被实际裁剪所使用。[LIAN84]给出Cyrus-Beck方法的代码。惟一的不同是裁剪在四维中，不是三维。

6.6.5 映射到一个视口

在规格化投影坐标系统（也称为三维屏幕坐标系统）中，输出图元被裁剪。为本节的讨论，我们假设规范平行投影视见体已经被用于裁剪（如果该假设不正确的话，透视投影M转换透视投影视见体到平行投影视见体）。因此所有保留下来的输出图元的坐标在视见体$-1 \leqslant x \leqslant 1$，$-1 \leqslant y \leqslant 1$，$-1 \leqslant z \leqslant 0$中。

PHIGS程序员指定一个三维视口（该视见体的内容被映射到该视口）。三维视口包含在单

位立方体$0\leqslant x\leqslant 1$，$0\leqslant y\leqslant 1$，$0\leqslant z\leqslant 1$中。单位立方体的$z=1$前平面被映射到可放到显示屏幕上的最大的正方形中。我们假设正方形的左下角在(0, 0)。例如，在水平分辨率为1024，垂直分辨率为800的显示设备上，正方形由位置在(p_x, p_y)的像素组成，$0\leqslant p_x\leqslant 799$，$0\leqslant p_y\leqslant 799$。通过抛弃其$z$坐标，单位立方体内的点被显示。因此点(0.5, 0.75, 0.46)可能会显示在设备坐标(400, 599)上。在可见面确定（第13章）时，每一个输出图元的z坐标被用来决定哪一个图元可见，哪一个被其他z值较大的图元隐藏。

在单位立方体内用式$x_{\text{v.min}}\leqslant x\leqslant x_{\text{v.max}}$等给定三维视口，于是从规范平行投影视见体到三维视口的映射可看成一个三步的过程。在第一步，规范平行投影视见体被平移，使它的角点
231~232 (－1，－1，－1)成为原点。这一过程受平移$T(1, 1, 1)$影响。接着，被平移的视见体缩放变换到三维视口的大小，缩放系数为

$$S\left(\frac{x_{\text{v.max}}-x_{\text{v.min}}}{2}, \frac{y_{\text{v.max}}-y_{\text{v.min}}}{2}, \frac{z_{\text{v.max}}-z_{\text{v.min}}}{1}\right)$$

最后，适当地缩放变换后的视见体由平移$T(x_{\text{v.}min}, y_{\text{v.}min}, z_{\text{v.}min})$平移到视口的左下角。因此，合成的规范视见体到三维视口的变换是

$$M_{\text{VV3DV}}=T(x_{\text{v.min}}, y_{\text{v.min}}, z_{\text{v.min}})\cdot S\left(\frac{x_{\text{v.max}}-x_{\text{v.min}}}{2}, \frac{y_{\text{v.max}}-y_{\text{v.min}}}{2}, \frac{z_{\text{v.max}}-z_{\text{v.min}}}{1}\right)\cdot \mathrm{T}(1, 1, 1) \qquad (6\text{-}44)$$

注意，这与5.5节推导的窗口到视口的变换M_{wv}类似，但不相同。

6.6.6 实现小结

在全部的观察变换中有两种通用的实现。第一个（在图6-34中已经描述，且在6.6.1节到6.6.3节中已经讨论过）适合于输出图元定义在三维并且应用到输出图元的变换从不产生一个负的W的情况。其步骤如下：

1) 扩展三维坐标到齐次坐标。

2) 应用规格化变换N_{par}或N_{per}。

3) 被W除并映射回三维（在一些情况下，已知$W=1$,所以不需要除法）。

4) 对平行投影或透视投影规范视见体（无论哪一个适合）进行三维裁剪。

5) 扩展三维坐标到齐次坐标。

6) 使用M_{ort}，式(6-11)，执行平行投影，或使用M_{per}，具有$d=-1$的式(6-3)，执行透视投影。

7) 使用式(6-44)平移和缩放到设备坐标中。

8) 被W除从齐次坐标映射到二维坐标；除法影响透视投影。

步骤6和步骤7使用一个矩阵乘法实现，且与图6-34中的阶段3和阶段4对应。

第二种实现观察操作的方法的需求产生于当输出图元定义于齐次坐标中且可能$W<0$时，或者当应用到输出图元的变换可能产生一个负W，或者当执行一个裁剪算法时。如6.6.4节所讨论的那样，其步骤如下：

1) 扩展三维坐标到齐次坐标。

233 2) 应用规格化变换N_{par}或N'_{per}（包括M，式(6-38)）。

3) 如果$W>0$，在式(6-42)定义的空间上对齐次坐标进行裁剪；否则在式(6-42)和式(6-43)定义的两个视见体上对齐次坐标进行裁剪。

4) 使用式(6-44)平移和缩放变换到设备坐标。

5) 被W除从齐次坐标映射到二维坐标；该除法影响透视投影。

6.7 坐标系统

在第5章和第6章已经使用了几种不同的坐标系统。在本节，我们总结所有的系统，并讨论它们之间的关系。还给出在各类参考文献和图形子程序包使用的同义词。图6-46显示坐标系统的接替关系，使用了本书通用的术语；因此，在任何特定的图形子程序包中，仅使用某几个坐标系统。我们已经为不同坐标系统选择名称以反映共同的用法，因此一些名称之间逻辑上并不是一致的。注意术语空间（space）有时作为系统（system）的同义词。

从图6-46左边与实际的显示设备距离最远的坐标系开始，每个物体被定义在一个**物体坐标系**中。PHIGS称之为**模型坐标系**；术语**局部坐标系**也常用。如我们将在第7章深入讨论的一样，经常有一个模型坐标系的层次。

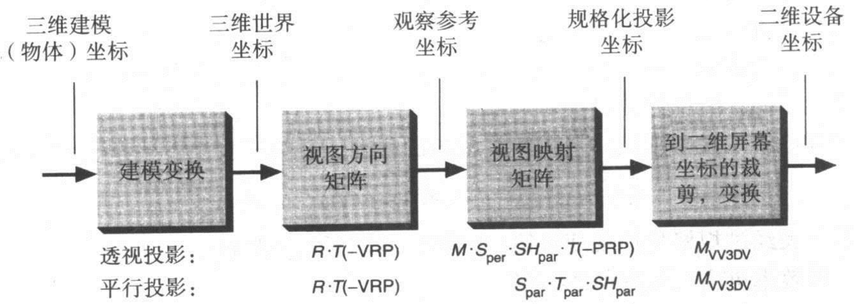


图6-46 坐标系统和它们彼此之间的关联。对于透视投影和平行投影，每个阶段的矩阵影响在该阶段应用的变换

物体被变换到**世界坐标系**中，在该系统中，一个场景或整个物体通过**模型变换**表示在计算机中。该坐标系有时被称为**问题坐标系**或**应用坐标系**。 234

观察参考坐标系被PHIGS用作为一个定义视见体的坐标系。它也称为(u, v, n)系统或(u, v, VPN)系统。Core系统[GSPC79]使用一个类似的但是未命名的左手系，该左手系被使用后，在原点用眼睛或照相机朝$+z$看，离眼睛越远，z值越大，x指向右边，y朝上。

其他图形包，例如Pixar公司的 RenderMan[PIXA88]，给观察参考坐标系加上约束，要求原点在投影中心，同时视图平面法线是z轴。我们称之为**眼睛坐标系**；RenderMan和某些其他图形系统使用**照相机坐标系**这一术语。参考6.6节，透视投影规格化变换的前三个步骤从世界坐标系转换到眼睛坐标系。眼睛坐标系统有时是左手的。

从眼睛坐标，我们接着到**规格化投影坐标系**或**三维屏幕坐标**，平行投影规范视见体的坐标系（和透视变换后透视投影规范视见体的坐标系）。Core系统称该系统为**三维规格化设备坐标**。有时，该系统被称为**三维逻辑设备坐标**。术语*规格化*一般意味着所有的坐标值或者在[0, 1]区间，或者在[−1, 1]区间，但是术语*逻辑*一般表示坐标值在其他预先指定的范围，例如[0, 1023]，该范围被典型地定义成对应于一些广泛使用的设备的坐标系统。在某些情况下，该系统不是规格化的。

从三维投影到二维产生我们所说的**二维设备坐标系**，也称为**规格化设备坐标系**，被[SUTH74a]称为**图像坐标系**，或者RenderMan称为**屏幕坐标系**。其他使用的术语包括**屏幕坐标**、**设备坐标**、**二维设备坐标**、**物理设备坐标**（与前面提到的逻辑设备坐标相反）。RenderMan称空间物理形式为**光栅坐标**。

不幸的是，对这些术语没有一个标准用法。例如，术语**屏幕坐标系**被不同的作者用来意指前面讨论的最后三个系统，覆盖了二维和三维坐标以及逻辑和物理坐标。

习题

6.1 试写一个接受观察定义的程序，并计算出N_{par}或N_{per}，同时显示这个房子（其坐标在图6-18中定义）。

6.2 写出用于平行投影和透视投影的三维裁剪算法的程序。

6.3 假设$F=-\infty$且$B=+\infty$，试证明：对于平行投影，先在三维裁剪然后投影到二维的结果
235 与先投影到二维再在二维裁剪的结果相同。

6.4 假设所有的物体在投影中心的前面，同时如果$F=-\infty$且$B=+\infty$,证明在透视投影规范视见体上对三维的裁剪然后进行透视投影的结果与首先透视投影到二维然后在二维裁剪的结果相同。

6.5 证明：S_{per}(6.6.2节)变换图6-42a的视见体为图6-42b的视见体。

6.6 写出以单位立方体进行三维裁剪的程序。将该程序推广到以任何长方体进行裁剪的一般情况，该长方体的表面垂直于各主轴。试问：该通用程序比适合于特定单位立方体的裁剪程序效率高还是低？解释你的答案。

6.7 试写出用规范透视投影视见体进行三维裁剪的程序，并推广到由下式定义的视见体：

$$-a\cdot z_v\leqslant x_v\leqslant b\cdot z_v,\quad -c\cdot z_v\leqslant y_v\leqslant d\cdot z_v,\quad z_{\min}\leqslant z_v\leqslant z_{\max}$$

这些关系代表经过规格化透视变换第1步到第4步以后的视见体的一般形式。试问，这两种情况哪一种效率更高？解释你的答案。

6.8 试写出用一般的六面视见体进行三维裁剪的程序，六个表面由下式定义：

$$A_ix+B_iy+C_iz+D_i=0,\quad 1\leqslant i\leqslant 6$$

比较下面的两种情况所需要的计算工作量：

a. 对每一个规范视见体进行裁剪。

b. 应用N_{par}，然后用单位立方体进行裁剪。

6.9 考虑三维空间中的一条直线，从世界坐标点$P_1(6, 10, 3)$到$P_2(-3, -5, 2)$，一个在区域$-z\leqslant x\leqslant z$，$-z\leqslant y\leqslant z$定义的半无限的观察四棱锥由下面平面界定：$z=+x$，$z=-x$，$z=+y$，$z=-y$。投影平面在$z=1$处。

a. 在三维空间裁剪该直线（用直线参数方程），然后将其投影到投影平面上。试问，在投影平面上裁剪后的端点是什么？

b. 将该线段投影到投影平面上，然后用二维计算对该投影进行裁剪。试问：在投影平面上裁剪后的端点是什么？

（提示：如果a和b的答案不一致，再试一下！）

6.10 当一个位于投影中心的“后面”的该物体用M_{per}进行投影，然后再进行裁剪时，则发生什么样的情况呢？你的答案应该证明为什么一般不能先投影再裁剪。

6.11 考虑带有一个旋转窗口的二维观察操作。设计一个规格化变换，将窗口变换成单位正方形。如图6-47所示，窗口是在VRC坐标系统由u_{min}, v_{min}, u_{max}, v_{max}指定的。试证明，这一变换与一般的三维 N_{par}相同，条件是投影平面是(x, y)平面，VUP有$-\sin\theta$ 的x分量和$\cos\theta$ 的y分量（也就是说，VUP到视图平面的平行投影
236 是v轴）。

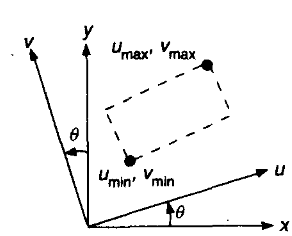


图6-47 一个旋转的窗口

6.12 将M_{per}作用于那些z坐标值小于0的点，其效果是什么？

6.13 设计并实现一套实用的子程序，它用来产生一个由基本变换R, S, T的任意序列组成的4 × 4变换矩阵。

6.14 画一个用来确定在创建图像时应用的投影的类型的判定树，将这个判定树应用到本章中三维投影的图中。

6.15 平行投影的规范视见体作为2 × 2 × 1矩形平行管道形状。假设单位立方体在正卦限位置，一个角在原点。

a. 找到该视见体的规格化N'_{par}。

b. 找到对应的齐次坐标视见体。

6.16 给出VRP在窗口中间时图6-18房子的前视图、顶视图和侧视图的观察参数。对每个视图，PRP一定不同吗？解释你的答案。

6.17 立体对是同一场景的两个视图，产生于略有不同的投影参考点，但是具有同样的视图参考点。设d是立体偏差，即两个参考点之间的距离。如果我们将参考点作为眼睛，于是d是我们眼睛之间的距离。令P是我们眼睛中间的点。给定P, d, VRP, VPN和VUP，推导两个投影参考点的表达式。 237

第7章 对象的层次结构和简单的PHIGS系统

图形软件包是一个应用程序和图形硬件间的中介。一个软件包可以只支持最基本的输出图元和交互设备，也可以提供非常丰富的功能。在第2章，我们介绍了相对简单和低级的SRGP软件包，并指出了其局限性。在本章，我们将介绍一个相对丰富也更复杂的标准图形软件包PHIGS（程序员的层次交互图形系统[⊖]）。诸如PHIGS和GKS（图形核心系统）这样的**标准软件包**是由国际或各国官方标准化组织制定的；GKS和PHIGS就是由ANSI（美国国家标准学会）和ISO（国际标准化组织）制定和发布的。这些标准的主要目的是使应用程序和软件开发过程具有可移植性。7.11.6节中进一步讨论的非官方工业标准也已证明了标准对于交互图形学的重要性。这些标准中包括了Silicon Graphics公司的OpenGL[NEID93]和Ithaca Software公司的HOOPS™[BASS90]。虽然在PHIGS PLUS、OpenGL及HOOPS之间有许多差别，但它们的相似之处要远多于它们的不同。如不讨论过程的API（应用程序接口）方面，至少在基本能力方面它们有更多的相似之处。读者在本章学习的许多内容，经过很小的变动就可以应用到其他软件包，因为它们都支持带变换的对象层次等功能。 239

这里描述的软件包实际上是PHIGS的一个子集，因而称为SPHIGS（Simple PHIGS，读“ess-figs”）。它保留了PHIGS的主要功能，但做了简化和修改以适应一些简单的应用。SPHIGS也包括了一些PHIGS PLUS扩展中的增强功能。我们设计SPHIGS的目的是用最简单的方式介绍有关概念，而不是要提供一个和PHIGS严格地向上兼容的软件包。不过，一个SPHIGS应用程序通过简单的修改就可以在PHIGS下使用。脚注说明了SPHIGS和PHIGS之间的一些主要区别，但总的来说，除非有特别的说明，否则SPHIGS的特点在PHIGS中都有。

在SPHIGS和集成的光栅图形软件包（如SRGP或X Window系统的Xlib软件包）之间有三个主要的区别。首先，为适应工程和科学应用的要求，SPHIGS使用三维的浮点坐标系统，并采用了第6章讨论的三维观察流水线。

其次，更大的不同是SPHIGS维持一个结构的数据库。所谓**结构**是一个基本图元、属性和其他信息的逻辑的集合体。程序设计人员可以通过少量的编辑命令修改数据库中的结构。SPHIGS保证屏幕图像是所存储的数据库内容的精确显示。结构不仅包含图元及其属性的定义，而且包括对从属的子结构的调用。这样，SPHIGS显示了类似编程语言中的过程的某些特点。特别是，就像由过程调用子过程，从而引入过程的层次一样，由结构调用其子结构，也就引入了结构的层次。当我们可以控制所调用的子结构的几何属性（大小、方向和位置）和外观（颜色、线型和线宽等）时，这种层次化的结构就显得有用。

第三个不同在于SPHIGS是在抽象的三维世界坐标系中，而不是在二维的屏幕坐标系中进行操作，所以它不支持对像素的直接操作。由于这些不同，SPHIGS和SRGP适应不同的需要和应用，正如我们在第2章所指出的那样，每一个软件包有它适用的地方，同时任何一个软件包

⊖ 本章中PHIGS这个术语，也包括PHIGS的扩展——PHIGS PLUS，PHIGS PLUS支持多面体、曲线和曲面等高级几何图元，也包括使用光照和明暗处理等的绘制技术，详见第12~14章。

都不能满足所有的需要。

因为其支持结构层次的特点，SPHIGS特别适合于基于构件和子构件的层次模型的应用；事实上，SPHIGS的结构层次可以看成是特殊功能的造型层次。因而在讨论利用SPHIGS进行几何造型的特点之前，我们在7.1节先讨论通常的几何造型。在7.2节至7.9节，我们讨论如何创建、显示和编辑SPHIGS的结构数据库。7.10节讨论交互，尤其是关联拾取。

7.1 几何造型

在自然科学和社会科学的教程中我们遇到过很多模型的例子。例如，大家可能熟悉原子的
240 波尔模型，在此模型中，电子围绕由中子和质子组成的原子核运动。其他的例子包括生物学中的非约束的指数增长模型，试图描述经济领域某些问题的宏观和微观经济学模型。模型是具体或抽象的实体的一些（无须全部）属性的表示。实体模型的作用是让实体的结构或行为可以被形象化和理解，并且提供一个可实验的手段，以及预测输入或变化对模型的影响。在自然科学、社会科学以及工程中的定量模型常常表达为方程组，可以通过改变独立变量、系数和指数的值进行实验。通常模型会对所描述的实体的结构和行为进行简化，以使模型易于形象化，或使由方程组表示的模型易于通过计算推导。

本书中，我们只讨论基于计算机的模型，特别是那些借助于图形学解释的模型。图形学可用于创建和编辑模型，获得模型参数的值，并使模型的结构和行为可视化。模型本身和创建模型并使之可视化的图形方法是不同的，模型，如人口模型，无须任何内在的图形属性。在通常的模型类型中，用到计算机图形学的模型有：

- *组织模型*：用层次的方法表示组织机构和分类，如图书馆的分类目录和生物学分类法。这些模型有多种有向图表示，如组织结构图表。
- *定量模型*：用方程来描述经济、金融、社会、人口、气候、化学、物理和数学系统。它们常用图表和统计图表描述。
- *几何模型*：有明确的几何定义的构件以及构件之间的关系构成的集合，包括工程和建筑结构、分子和其他化学结构、地理结构以及车辆等。这些模型通常由结构图或模拟现实的合成图像表示。

计算机辅助建模可以帮助医药研制人员模拟针对某种疾病的新混合物的化学行为，帮助航空工程师预测超音速条件下机翼的变形，帮助飞行员学习驾机飞行，帮助核反应堆专家预测各种设备发生故障的影响并提出适当的补救措施，以及帮助汽车设计师测试在撞车时乘客车厢的完整性。在这些以及其他的更多的例子中，利用模型进行实验比用实物进行实验更容易更经济也更安全。事实上，在许多情况下，如航天飞机飞行员的训练和核反应堆安全性的实验，建模和仿真是惟一可行的获取系统知识的方法。正是因为这些原因，计算机建模正在逐渐代替传统
241 的方法，如风洞实验。工程师和科学家可以利用数字的风洞、显微镜、望远镜等进行许多实验。这种利用数值分析的方法进行模型的模拟和动态显示的方法正成为科学中的一种新范例，逐渐代替传统的理论和物理实验分支。建模和模拟的成功取决于一个好的模型和输入，在建模中尤其要避免发生所谓的“输入垃圾产出垃圾”的情况。

模型并不一定需要包含内在的几何数据；像组织机构模型这样的抽象模型并不是面向空间的。尽管如此，许多这样的模型能被几何地表示，例如，组织机构能用机构图表示，临床药剂评价的结果可以用直方图表示。即使是表示内在的几何物体的模型，也未必要采用图形表示或模型的视图来描述。例如，我们可以选择用多面体或是用曲面的集合来表示机器人，同时我们也能规定诸如以什么视角、以何种投影关系以及要求达到怎样的真实效果，来对其进行真实感的描述。我们

也可以选择以图形的方法来显示模型的结构或者模型的行为，比如，我们也许需要显示一个超大规模集成电路芯片上的物理电路，同时也需要显示其作为输入和时间函数的电子和逻辑行为。

7.1.1 几何模型

几何模型或图形模型由于使用内在的几何性质描述构件，很自然地可用图形表示。几何模型可表示为如下成分：

- 构件的空间位置分布和形状（即实体的*几何属性*）和其他影响外表的属性，如颜色。
- 构件间的连接关系（即实体的结构与*拓扑属性*），这些连接属性可以抽象定义（如用网络的关联矩阵或层次关系的树结构）；也可以由其内在的几何定义（如集成电路的通道的维数）。
- 和构件相联系的由应用所定义的数据值和属性，如电子文本和描述文字。

和几何模型相关的是其处理算法，如对离散电路模型的线性电路分析、对机械结构的有限元分析以及原子模型的极小能量方法。

直接被存储在模型中的部分和在分析显示前通过计算得到的部分之间通常有一个平衡，即通常所说的空间-时间的平衡。例如，在一个计算机网络的模型中，可直接存储所用的连接关系，也可以每次当需要一个新的视图时，通过连接矩阵用一个简单的图布局算法计算得出。为了满足分析和显示的要求，一个模型需要保持足够的信息，但是信息表达的格式和编码技术的选择，取决于应用和时间-空间的平衡。 242

7.1.2 几何模型中的层次

几何模型通常具有一个由自底而上的构造过程决定的层次结构：构件被作为构成高一层次实体的部件（building block），同时，新的实体又作为更高层次的实体的部件。如同大型的编程系统一样，其层次结构很少是以严格的自底向上或自顶向下的方法建成，其根本的是最后的层次结构而非其建构的过程。物体具有层次结构的属性非常普遍，因为几乎没有实体是完全独自为一体的，一旦我们将一个物体分为其零件的集合，我们已经至少创立了一个两层的层次结构。在一些特殊情况下，每个物体只被其上一层次的物体包含一次，这种层次结构可用树来表示，其中每个物体是一个节点，物体间的包含关系是连接节点的边。在更常见的情况下，物体可能被包含多次，这种情况的层次结构可用*无环有向图*（DAG）表示。图7-1是一个物体层次结构的简单例子，表示一个基本的机器人透视图；图7-2a用DAG显示了机器人的结构。我们可以复制多重包含的物体，把无环有向图转化为图7-2b所示的树。为表示方便，单向的箭头被省略，因为节点间的关系可以通过树中节点的相对位置表示，在上面的节点包含在下面的节点。

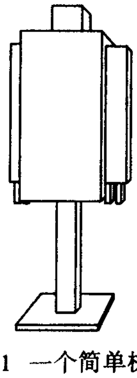


图7-1 一个简单机器人的透视图

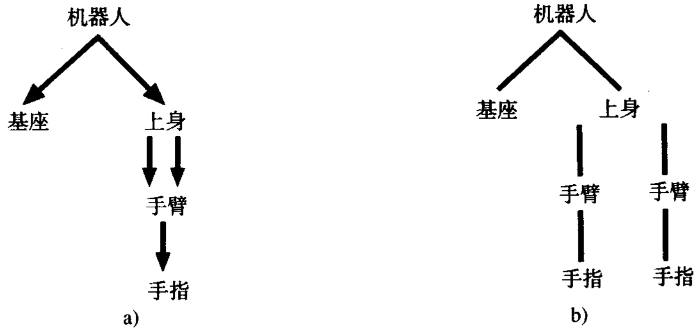


图7-2 a) 机器人构件的层次结构的无环有向图，b) 机器人层次结构的树表示

机器人由一个在基座及其上可转动的上身组成。上身由头和躯干组成，头可绕躯干转动；躯干还和两个独立的手臂相连，手臂可分别绕过肩膀的水平轴转动。手臂由一个固定的部分即手部和一个与手保持平行并可滑动的手指组成以完成基本的抓取动作。这样，一个手指构件在手臂中被包含一次，而在一个躯干中则包含两个手臂构件。在本章中我们将讨论这个机器人的创建，它的形状以正投影视图的形式显示在7-5b给出的屏幕的2~4窗口中。

虽然具有层次结构的物体既可以由几何图元组成，也可以由其低层次的包含物组成，但表示机器人层次结构的有向图或树只显示了其和所包含的低层物体之间的关系。这如同在高级的过程语言中，过程的层次关系图用来显示程序的调用结构一样。重要的是，一个复合对象如何被构造成层次结构是由设计人员决定的。例如，机器人可以以两层的层次结构来构造，根对象由作为几何图元的基座、头、躯干，以及同样由几何图元表示的相连的两支手臂构成。

许多系统（如计算机网络和化学设备）可以用网络图表示，其中的对象不仅被多次地包含，而且还可以拥有任意的连接关系。这样的网络可以表示为可能包含环的图，但在层次结构中，子系统多次出现时，系统依然能显示对象间的包含层次。

243 为了简化构造复杂对象及其模型的任务，我们通常用由应用所决定的原子构件作为基本的部件。在二维中，这些构件通常用标准符号形状的塑料模板或计算机绘图模板来绘制。在绘图程序中，这些形状相应地由基本几何图元，如直线、长方形、多边形、椭圆、圆弧等构成。在三维中，圆柱体、平行六面体、球体、锥体、旋转曲面等常被用来作为部件。这些三维形体部件可以由低层次的几何图元（如三维多边形）来定义；在这种情况下，光滑曲面以牺牲其精度为代价，可以由多个多边形来逼近。在可以直接处理自由曲面和实体的高级几何造型系统中，诸如参数多项式曲面这样的形状以及诸如圆柱体、球体、圆锥体这样的实体，可以将其自身作为图元，并可以直接解析地定义，从而不损失精确性（参见第9章和第10章）。本章中，我们将用“对象”的概念描述由其自身的模型坐标系以几何图元或低层的物体定义的构件，它不仅包含几何数据，也包含相关的应用数据。这样，一个对象就是一个作为组成部分的形状以及其所用的数据。

于是，我们可以构建适于多种用途的层次结构：

- 用模块的方法建立复杂的对象，通常可以通过反复地调用具有不同几何属性和外观属性的部件来实现。
- 提高存储的效率，因为只存储反复使用的对象的索引，而不必每次存储完整的对象定义。
- 允许方便的更新传播，因为在一个作为部件的对象定义中的修改将自动传播到所有使用这个部件的高层次的对象中去（既然它们已代表一个已更新的版本）。这和编程语言中的过程层次类似，对一个过程体的改变将会反映到那个过程的所有调用中。

应用程序可以使用多种技术来编码层次模型。例如，一个网络或关系数据库可以被用来存储对象的信息以及对象之间关系的信息。另一种更有效的方法，是由应用程序维持一个自定义的链表结构，包括存储对象的记录和描述关系的指针。在一些模型中，对象之间的关系也是对象，它们必须在模型中以数据记录的形式来表示。另外一种方法是使用面向对象的数据库[ZDON90]。面向对象的编程环境，如SmallTalk[GOLD83]、MacApp[SCHM86]和C++[STRO91]正在被越来越多地应用于图形应用程序中，用于存储几何对象的模型信息。

1. 相互关联

在许多网络中，对象被置于特定的位置（或由用户交互或由应用程序自动决定），然后被相互关联。这种关联可以是抽象的，从而是任意形状的（在层次结构图或网络图，如组织机构
244 图或工程进度图中）；也可以是本身具有意义的几何结构（如超大规模集成电路芯片）。如果

是抽象的关联，我们可以用多种标准协定得出层次结构图或网络图，并且我们也能用诸如线型、线宽或颜色等属性来表示各种不同的关系（如组织机构图中的虚线表示）。对于那些形如集成电路中半导体部件间的电路连接情况的关联，本身就是对象。抽象的和非抽象的关联通常都被**约束**在水平或垂直的方向上（有时称为**曼哈顿**布局模式）以简化可视化和物理构建过程。

2. *对象层次间的参数传递*

作为部件调用的对象必须置于其父对象的适当的位置，为了适应此要求，通常必须改变对象的大小和方向。齐次坐标矩阵用于第5章的图元变换以及第6章的视见体规格化；在层次结构的模型中，对低层对象使用缩放、旋转和平移矩阵是常见的操作。Sutherland首先在Sketchpad系统[SUTH63]中使用这种图形造型的能力，并使用了**宿主**（master）这个术语来表达对象的定义，**实例**（instance）表达调用几何变换的情况。如在4.3.3节中讨论的那样，在使用**层次结构显示列表**的图形系统（又被称为**结构显示文件**）中，在硬件中实现了宿主实例的层次结构，使用子循环调用和浮点算术单元来实现变换。为了区分用于视见体规格化的变换和用于建构层次结构对象的变换，我们通常将后者称为**模型变换**。在数学上，模型变换和规格化变换间没有区别。

和过程层次结构相比，我们有时称一个父对象"调用"层次结构中的一个子对象，并且将父对象坐标系中的比例、方向、位置等"几何参数"传递给子对象。我们可以看到，像SPHIGS这样的支持对象层次结构的图形软件包，能对图元的顶点和实例化的子对象的顶点做存储、组合和使用变换矩阵。但在7.5.3节我们将看到SPHIGS的参数传递机制并不像面向过程语言那样普遍。

7.1.3 模型、应用程序和图形系统间的关系

至此，我们大致地讨论了模型，并着重讨论了层次结构的几何模型和模型变换。在讨论SPHIGS前，让我们简要地回顾一下图1-11显示且后来在图3-2中进一步解释的图形的概念模型，以进一步揭示模型、应用程序和图形系统间的内在关系。在图7-3中，应用程序分为五个子系统，记为a)到e)： 245

a) 通过增加、删除和替换模型中的信息来建立、修改和维护模型。

b) 遍历（扫描）模型以提取用于显示的信息。

c) 遍历模型以提取信息用于分析模型的行为和性能。

d) 显示信息（如绘制几何模型、输出分析结果）和用户界面工具（如菜单、对话框）。

e) 执行不直接用到模型和显示的各种应用任务（如内务处理）。

术语**子系统**并不一定指编码的主要模块——少量的调用或一个短的过程都足以实现一个子系统。并且，一个子系统可以贯穿应用程序的始终，而不是集中在一个独立的程序模块中。图7-3只是简单地显示了逻辑构件而不是程序结构构件，并且不同于建立、修改、分析或显示模型的过程，至于调用的是模型的特定模块还是模型维持代码的特定部分，并不总是清楚的。这可用以下例子说明，比如，电路分析模块是模型定义的一部分，因为它描述了模型的行为。对于一个使用传统的过程语言（如C和Pascal）的程序员来说，如果认为模型主要包括数据，图7-3会更好理解，而熟悉面向对象的编程语言的人认为模型同时包括数据和过程会更自然一点。

在许多应用程序中，特别是工业应用中，存在着"80/20"规则：程序的主要部分（80%）处理实体的建模，只有少数部分（20%）处理模型的显示。换句话说，在许多如CAD那样的应用程序中，模型的图形表达只是一种达到目的的手段，用于分析、物理建构、数控加工以及其他的后处理。当然也有许多应用就是用于图形本身，如画图、绘图、电影和视频制作、飞行模拟场景的动画等，这其中除绘画外，都需要一个模型以便于产生绘制的图像。简言之，大多数的图形涉及模型（和模拟），说"图形学就是建模"是有道理的。第9章和第10章将讨论这个重要的内容。

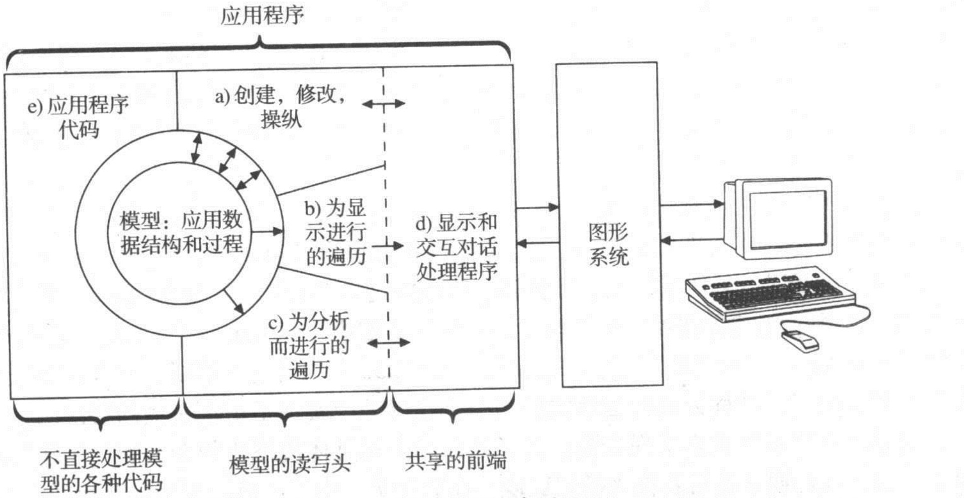


图7-3 应用模型及其读写

7.2 保留模式图形包的特点

在讨论应用软件、应用模型和图形软件包的作用时，我们先看一下图形软件包的功能和模型改变后的影响。如第2章中提到的，SRGP在**立即模式**下进行操作，不能保持传递给它的图元和属性的记录。这样，对一个应用对象的修改或删除需要除去屏幕上的所有相关信息，然后有选择地改变或全部重新生成屏幕信息，这些都需要应用程序从模型中重新定义图元。另一方面，PHIGS在**保留模式**下工作：它保持一个所有图元和相关信息的记录，以允许其后的编辑和自动更新显示，这样它就减轻了应用程序的负担。

7.2.1 中央结构存储库及其优点

PHIGS在一个叫作**中央结构存储器**（CSS）的特殊用途数据库中存储信息。PHIGS中的**结构**是一些**元素**的序列，这些元素包括图元、外观属性、变换矩阵和对从属结构的调用，其目的是定义一致的几何对象。这样，PHIGS有效地存储了一个特殊用途的造型层次结构，并把造型变换和其他属性作为参数传递给从属结构。请注意CSS造型的层次结构与存储宿主-实例的层次结构的硬件分层显示列表的相似性。事实上，PHIGS可以看成一个与设备无关的层次化显示列表的软件包规范；虽然给定的实现为特定的显示设备而被优化，但编程人员不需要关心其细节。尽管许多PHIGS
246~247 的应用是纯软件的，但大多数安排是在软件上做CSS操纵，并将软件和硬件的组合用于绘制。

和显示列表一样，CSS复制存储在应用程序的通用模型/数据库内的几何信息，以实现快速的显示遍历，即用于计算模型的新视图的遍历。CSS的主要优势在于当应用更新CSS时，能自动进行快速的屏幕刷新。仅这个属性就使得在应用数据库和CSS间复制几何数据变得物有所值，尤其当PHIGS应用程序使用独立的处理器作为“遍历引擎”以使CPU减轻显示所需遍历的计算负担的时候。诸如改变变换矩阵等较小的编辑功能，也能在PHIGS中有效地完成。

CSS的第二个优势是可以自动地关联拾取：软件包确定由用户拾取的图元在层次结构中的实体和位置（见7.10.2节）。关联拾取功能展示了将使用频率高的功能纳入基本软件包的常用技巧。

CSS的第三个优势是其编辑功能，与层次建模的特点相结合，使用户能容易地创建各种动态效果，如运动过程的动力学，其中时变的变换可用于在父对象中对子对象进行缩放、旋转和定位。例如，我们可以用施加在子结构上的旋转表示我们的简单的机器人模型的每一个关节

（如手臂是上身的一个可旋转的子结构），并通过编辑一个旋转矩阵来动态地旋转手臂。

7.2.2 保留模式软件包的局限性

尽管CSS（作为一个针对显示和快速增量式刷新的特殊用途的实体）实现了一些常用的造型操作，但是它不是对每一种造型的应用都是必需的和充分的。CSS不总是必需的，因为应用程序在模型被修改时能自行完成屏幕刷新，能自行完成关联拾取（即使需要可观的工作量），并且能通过定义对象、传递变换以及其他参数的过程来实现对象的层次化结构；CSS通常是不充分的，因为在大多数的应用中，仍然需要单独地来建立和更新应用的数据结构，以记录每个应用对象的所有相关的数据。这样，将存在一个所有几何数据的副本，并且这两个表示间必须保持同步。因为这些原因，一些图形软件包不需要进行任何类型的结构存储，就可支持浮点坐标以及规格化的二维和三维观察功能。这种立即模式的软件包的合理性在于维持CSS所需的代价，因为通常应用程序本身都维护一个应用模型，这个模型已足以满足屏幕刷新的要求。

对于那些在相继的图像之间有很大的结构改变的应用，保留模式软件包就不能满足需要。例如，在“数字风洞”中进行机翼分析时，其表面是由三角形网格表示的，当机翼受空气动力作用时，其三角形网格的大多数顶点将发生位置改变。在这种情况下编辑结构数据库就没有意义，因为每个新图像中的大多数数据都发生了变化。事实上，我们并不推荐编辑PHIGS的结构 248
数据库，除非被编辑的元素的数目远小于整个被显示的网格。PHIGS所提供的编辑工具只是最基本的，比如，可以很方便地改变一个模型变换；但是要改变一个多边形的顶点就需要先删除原多边形然后再重新定义新的版本。通常情况下，应用程序会为显示遍历而不是为大规模编辑做优化，因为显示遍历是最常用的操作。另外，任何情况下应用模型都必须要更新，而更新一个数据库总是比更新两个数据库更方便快捷。

由于这些局限性，一些PHIGS的实现提供了立即模式输出的功能，或者提供一种混合模式，该模式图元用立即模式指定而可能与保留的图元联合在一起。

7.3 定义和显示结构

上一节已经讨论了PHIGS和SPHIGS的基本的属性，本节中，我们开始详细描述SPHIGS软件包，除非特别注明，否则所有讨论的内容都适用于PHIGS。SPHIGS结构上允许的操作包括以下几项：

- 打开（启动编辑）和关闭（终止编辑）。
- 删除。
- 插入**结构元素**（三种主要类型的结构元素是图元、属性（包括定义模型变换的属性）和调用子结构的元素）。一个元素是每当一个**元素生成过程**被调用时生成并被插入当前结构中的一个数据记录，它存储了那个过程的参数。
- 删除结构元素。
- **提交**显示（可以类比于在布告栏提交照片），由定义如何将浮点坐标映射到屏幕坐标的**视图操作**决定。

7.3.1 打开和关闭结构

创建一个结构，比如构造一个形成图7-2中机器人手臂的图元和属性的集合，我们把对元素生成函数的调用与以下调用归为一类：

```
void SPH_openStructure ( int structureID );
void SPH_closeStructure();
```

这些函数对结构进行的操作，类似于标准打开和关闭文件命令对磁盘文件进行的操作。但和磁盘文件所不同的是，每次只能打开一个结构，并且在打开时定义的所有元素都存储在此结构中。
249 一旦关闭，结构可重新被打开以供编辑（见7.9节）。

结构有另外的两个特点：首先，图元和属性可以仅仅被确定为结构的元素。对于一个结构中可以存储多少个元素并无规定；结构既可以是空的也可以包含任意多个元素，仅受存储空间的限制。当然，构成一个结构的所有元素通常应该是定义了一个对象的逻辑上相互关联的集合。

其次，结构的标识号是整数，由于这些标识通常只由应用程序使用，而非交互用户使用，所以编程人员可以用符号常量表示这些标识，而一般并不需要用字符串这样的形式。整数标识可以允许在应用程序数据结构中的对象和相应对象的结构标识间建立方便的映射。

7.3.2 定义输出图元及其属性

生成输出图元元素的函数和在SRGP中的看起来很相似，但两者间有一些重要的差别。首先，点由三个双精度的坐标（x, y和z）定义，其次，这些函数把元素放在CSS中当前开放的结构中，而不是直接改变屏幕的图像——显示结构是一个独立的操作，将在7.3.3节中讲述。在本章，**图元**这个术语是三个相关实体的简称：元素生成函数（如SPH_polyLine），由该函数生成的结构的元素（如polyLine元素），以及在对中央结构存储器的显示遍历过程中，执行元素所产生的显示图像。SPHIGS通过模型变换和视图操作转换图元的坐标来实现图元元素的**执行**，这些操作包括将坐标裁剪、转换到视见体，然后**光栅化**（转换到像素）。SPHIGS中的属性比SRGP定义得更细，即每个图元都有其自己的属性。这样，像颜色和线型这样的属性是分类的，以便程序员可以在保留多边形和文本颜色的同时重置直线的当前颜色。

1. 图元

SPHIGS比SRGP有更少的图元，因为与一些SRGP的图元（如椭球体）等价的三维实体在实现时，尤其在变换、裁剪和扫描转换中，是耗费计算时间的。

大多数SPHIGS的图元和SRGP相应的图元具有相同的定义方法（除了点是三维的情况）：

```
void SPH_polyLine ( int vertexCount, pointList vertices );
void SPH_polyMarker ( int vertexCount, pointList  vertices );
void SPH_fillArea ( int vertexCount, pointList  vertices );
   /* 和SRGP_polygon相似 */
void SPH_text ( point origin, char *str );
   /* 不是完整的三维，参见7.7.2节 */
```

注意，SPHIGS不验证填充区域（或面片，下面将进行描述）是否是平面的，如果不是平面的
250 将会产生不可预知的结果。

现在，我们考虑图7-4所示的简单房屋的定义。我们可以通过把每个面（也称为**面片**）定义为一个填充区域在SPHIGS中描述房屋，以不必要地重复定义和存储（在CSS中）每一个共享的顶点为代价。这种重复还会减慢显示的产生速度，因为视图操作的计算必须对每一个顶点处理多次。利用共享顶点的间接面片索引可以大大提高存储和处理时间上的效率。我们把一个多面体看成面片的集合，每个面片由一个顶点索引表表示，每个索引是一个指向顶点表的指针。我们可以用以下的记号描述一个多面体的说明：

Polyhedron = {VertexList, FacetList}
VertexList = {V1, V2, V3, V4, V5, V6, V7, V8, V9, V10}
V1 = (x_1, y_1, z_1), V2 = (x_2, y_2, z_2), …
FacetList = {front = {1, 2, 3, 4, 5}, right = {2, 7, 8, 3}, … bottom = { … } }

SPHIGS提供这种有效的形式来定义**多面体**图元。在SPHIGS术语中，多面体是面片的集合，

它可以封闭也可以不封闭一个体。在一个封闭的多面体中，如我们的房屋，每个顶点通常至少被三个面片所共享，所以利用间接方法进行定义的效率是很高的。多面体的外观也受到填充区域的同样属性的影响。

多面体元素生成过程中的面片表是以存储面片描述的连接集合的整数（SPHIGS中的类型“vertexIndexList”）数组形式表示的。每个面片描述是$V+1$个整数的序列，其中V是面片包含的顶点数，前V个整数表示顶点表的索引，最后一个整数(-1)是面片的结束标志。这样，我们可以通过发送数组1, 2, 3, 4, 5, -1, 2, 7, 8, 3, -1, …来描述房屋的面片（通过函数的第四个参数，下面将进行描述）：

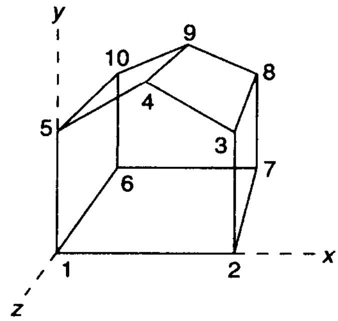


图7-4 一栋由点集和面片集定义的简单房屋

```
void SPH_polyhedron
    ( int vertexCount, int facetCount, pointList vertices, vertexIndexList facets);
```

注意，SPHIGS的绘制算法要求区别一个面的两面，即朝内的和朝外的。这样，一个面片的顶点应该以逆时针顺序（右手规则）定义，视点位于面片外侧。⊖

作为一个简单的例子，下面的C程序创建了由单个多面体组成的结构，对图7-4的房屋进行造型：

```
SPH_openStructure (HOUSE_STRUCT);
    SPH_polyhedron (10, 7, houseVertexList, houseFacetDescriptions);
SPH_closeStructure();
```
251

基本上，SPHIGS只支持多边形几何图元，更高级的三维造型图元将在以后讨论——第9章的多项式平滑曲线及曲面和第10章的实体图元。

2. 属性

程序7-1所列出的函数生成属性元素。在显示遍历过程中，属性元素的执行按以下模式改变属性的值：新的值将一直起作用，直到被显式地改变。如同在下一节和7.7节中讨论的那样，显示遍历过程中，属性和图元是联系在一起的。

程序7-1 生成属性元素的函数

```
polyLine:
    void SPH_setLineStyle( style CONTINUOUS / DASHED / DOTTED / DOT_DASHED );
    void SPH_setLineWidthScaleFactor( double scaleFactor );
    void SPH_setLineColor( int colorIndex);

fill area and polyhedron:
    void SPH_setInteriorColor( int colorIndex );
    void SPH_setEdgeFlag( flag EDGE_VISIBLE / EDGE_INVISIBLE );
    void SPH_setEdgeStyle( style CONTINUOUS / DASHED / DOTTED / DOT_DASHED );
    void SPH_setEdgeWidthScaleFactor( double scaleFactor );
    void SPH_setEdgeColor( int colorIndex );

polyMarker:
    void SPH_setMarkerStyle( style MARKER_CIRCLE / MARKER_SQUARE / ... );
```

⊖ SPHIGS要求每个面片的一面被定义为朝外，即使这个多面体的面不封闭。另外，我们认为朝内的面是不可见的。

```
    void SPH_setMarkerSizeScaleFactor( double scaleFactor );
    void SPH_setMarkerColor( int colorIndex );

text:
    void SPH_setTextFont( int fontIndex );
    void SPH_setTextColor( int colorIndex );
```

填充区域的属性不同于在SRGP中的属性，填充区域和多面体图元都有分别单独定义了属性的内部和边界。内部只有颜色属性，而边界有颜色、线型、宽度等属性。另外，边界的可见性可以通过边界标志属性加以关闭，这一点对多种绘制模式都有用，这将在7.8节中讨论。

线或边界宽度以及标记大小是以非几何的方式定义的：它们不是通过世界坐标系单位定义的，因而不支持几何变换。因此模型变换和观察操作可以改变一条直线的显示长度，但不能改变其宽度。类似地，在不连续线型中的长划线的长度也是和作用于其上的变换相独立的。然而不同于SRGP的是，像素不作为测量的单位，因为其大小是设备相关的。一般是对每一设备设置一个标称宽度/大小，从而使得在每个输出设备上对宽度/大小单位有基本相同的外观。
252 SPHIGS的应用规定的单位为该标称宽度的倍数（非整数）。

SPHIGS因为三个原因而不支持图案：首先，SPHIGS保留图形以在二值显示系统中模拟彩色的光照效果；其次，对大多显示系统而言，实现彩色系统图案区域的光滑光照效果需要太多的计算量；最后，PHIGS中，称为“**剖面线**”的几何图案即使在有实时变换硬件的显示系统中也是非常耗时间的。

7.3.3 提交结构进行显示遍历

SPHIGS记录了在CSS中新生成的结构，但直到应用程序提交了服从特定观察规范的结构后，才显示这个新的结构⊖。SPHIGS然后在CSS中对结构元素进行**显示遍历**，从第一个到最后一个按顺序执行每个元素。如果图元的一部分在视图中可见，则图元将显现在屏幕图像上。执行一个属性元素（包括几何变换和外观属性）将改变存储在状态向量（**属性遍历状态**）中的属性集，该属性集将模式地作用于其后遇到的图元上。这样，属性以显示遍历的顺序作用于每个图元。

下面的函数在由SPHIGS内部维护的提交结构表中添加一个结构：

```
void SPH_postRoot ( int structureID, int viewIndex );
```

术语根表示在提交一个调用子结构的结构*S*时，我们实际上提交的是一个以*S*为根的层次化的DAG，称为**结构网络**。即使提交的结构不调用子结构，我们也称之为根，所有被提交的结构都是根。

*viewIndex*参数选择视图表（将在下一节讨论）中的项，这个项规定了结构的图元的坐标如何映射到屏幕的整数坐标空间。

我们可以通过删除CSS中的结构（或元素）的方式从屏幕上删除一个对象的图像（见7.9节），也可以通过使用SPH_unpostRoot函数从提交的根表中去除特定的根，而不从CSS中删除结构：

```
void SPH_unpostRoot ( int structureID, int viewIndex );
```

⊖ PHIGS和SPHIGS提交的结构显示的方式完全不同。PHIGS通常的机制是，视图规范是一个结构元素，从而在显示遍历时，可以像其他元素一样被改变和编辑。许多现有的PHIGS实现也支持SPHIGS型简单的结构提交机制。

7.3.4 观察

1. 人造照相机

正如第6章中所讲的那样，把三维软件包看成一个人造照相机，来获得由几何定义了的对象所占据的三维世界的“快照”。生成一个结构等价于把一个对象定位于照相空间，提交一个 253 结构可以类比为激活预先设置在某个场景中的照相机对物体进行拍照并将快照提交于布告栏。我们可以看到，每当场景中任何对象发生改变时，我们的照相机都将自动产生一个更新的图像以代替旧的。为了产生动画效果，快速地显示多幅静止图像，就如同一个照相机一样。

继续我们的比喻，让我们来考虑合成的图像是如何产生的。首先，照相机操作者必须定位照相机并调整其方向。其次，必须决定哪些场景应该被显示：例如，是对象近景的特写还是远距离的全景。接下来，摄影者必须决定做一个多大的照片以便提交在布告栏中。最后，要决定将照片提交在布告栏的什么位置。在SPHIGS中，这些原则在视图中表示，包括观察操作的定义，该操作的视口定义了照片的大小及其在布告栏的位置。并不是结构数据库中的所有对象都用同样的“照相机设置”来显示。事实上，我们很快会看到，在一个布告栏中可以有多个视图。

2. 视口

如同上一章讨论的，视口在规格化投影坐标（NPC）系统中定义了一个平行六面体，在观察参考坐标（VRC）中定义的视见体的内容被映射到这个区域内。由于NPC系统以一种固定的方式映射到物理设备的整数坐标系，因此视口也定义了图像将显示在屏幕的什么位置。三维的NPC系统以下面的方式映射到二维的屏幕坐标系：NPC单位立方体中位于(0, 0, 0)和(1, 1, 1)的两个对角点分别对应于屏幕上能表示的最大正方形顶点，而z坐标将直接被忽略。例如，一个显示设备的分辨率为水平1024竖直800，一个NPC下点$(0.5, 0.75, z)_{NPC}$对应设备坐标系下的$(511, 599)_{DC}$。为增强可移植性，应用程序不应使用位于单位立方体外的NPC系统空间；通常，为了充分利用非正方形屏幕的优势，为实现可移植性所牺牲的代价是值得的。

3. 视图表

SPHIGS维持一个视图表，其表项的数量与具体实现相关。每个视图包含一个称为**视图表示**的视见体和视口的定义，以及一个被提交的根列表（初始为空）。视图表中的第0项定义了一个**默认视图**，它是一个如图6-19b所示的视见体，前后平面分别位于$z = 0$和$z = -\infty$处。其默认视图的视口为NPC单位立方体。

视图表中所有项的视图表示（除视图0）都可以用以下过程来编辑：

```
void SPH_setViewRepresentation(
    int viewIndex, matrix_4x4 voMatrix, matrix_4x4 vmMatrix,
    double NPCviewport_minX, double NPCviewport_maxX,
    double NPCviewport_minY, double NPCviewport_maxY,
    double NPCviewport_minZ, double NPCviewport_maxZ );
```
254

两个4 × 4的齐次坐标矩阵是第6章所述的视图方向矩阵和视图映射矩阵，它们由程序7-2所示的函数生成。

4. 多视图

在提交时定义的视图索引指向一个特定的NPC视口，该视口描述了在屏幕（布告栏）上出现结构图像（照片）的位置，如同人们可以在布告栏上提交多幅照片一样，一个应用程序可以将屏幕分成几个视口。多视图的用途在许多方面都很强大。我们通过以不同的视图提交，可以在单个屏幕上不同的区域内同时显示几个不同的结构。在图7-5a中，我们示意性地表示一个视图表，只显示了指向每个视图的被提交的结构网络列表的指针。我们可以看到有一个视图显示了一个街道

的场景，有三个独立的视图显示了一个机器人。机器人的结构被显示了三次，每次有不同的视图索引。图7-5b显示了结果的屏幕图像。机器人的多个视图不仅可以具有不同的视口定义，也可以具有不同的视见体定义。

程序7-2　用于计算观察变换矩阵的实用程序

```
/* 建立UVN观察参考坐标系 */
void SPH_evaluateViewOrientationMatrix( point_3D viewRefPoint,
        vector_3D viewPlaneNormal, vector_3D viewUpVector,
        matrix_4x4 *voMatrix );

/* 建立视体并描述它怎样映射到NPC空间 */
void SPH_evaluateViewMappingMatrix(
    /* 首先，我们在VRC中定义视体 */
        double umin, double umax, double vmin, double vmax,
        int projectionType,           /* PARALLEL / PERSPECTIVE */
        point_3D projectionReferencePoint,              /* 在VRC中 */
        double frontPlaneDistance, double backPlaneDistance,  /* 裁剪平面 */
    /* 然后，我们指定NPC视口 */
        double NPCvp_minX, double NPCvp_maxX,
        double NPCvp_minY, double NPCvp_maxY,
        double NPCvp_minZ, double NPCvp_maxZ,
      matrix_4x4 *vmMatrix );
```

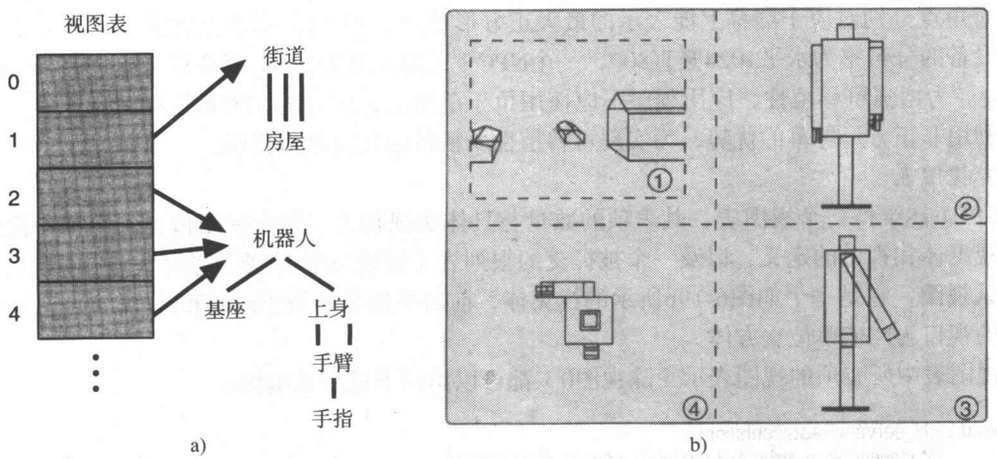


图7-5　共享同一屏幕空间的多视图。a)视图表的示意图，表中的每一项都指向一个提交给该视图的根列表；b)结果图像，虚线框定的视口范围和圆圈中的数字显示了视口和与之对应的视图索引

前述内容都表明每个视图至多有一个被提交的结构，但事实上，任意数目的根都能提交给单一的视图。因此，我们可以通过把所有根提交给一个视图来在一个统一的视图中显示不同的根结构。这种情况下，我们可比喻为用照相机拍摄一张包含许多物体的场景的合成快照。

视口不同于实际的快照或窗口管理器的窗口的另一个特性在于它是透明的。在实际应用中，许多程序为避免重叠而将视口平铺开，但有时叠放也有其优势。如我们可以合成两个由不同的观察变换产生的或在不同测量单位下显示的图像，这样，我们就可以在建立一个机器零件的近视图时插入一个整个机器的小图（供参考）重叠在大的零件图上。（我们可以选择近视图中只

有背景的区域。)

为刷新屏幕，SPHIGS通过以视图索引值递增的顺序遍历在视图表中的每个视图所提交的根来显示所提交的元素网络，通常从视图表中的视图0开始。这样，一个提交给视图*N*的对象图像将比一个提交给索引值小于*N*的视图的对象图像有更高的**显示优先级**，因而可能遮盖住它们。当然，只有当视口实际上重叠时，它们间的顺序才是十分重要的。(注意，这种平凡的视图优先系统不如PHIGS复杂，允许应用程序显式地指定视图优先级。)

注意，一个应用程序能产生许多独立的世界坐标系（WC）空间，并能使用任意所需的测量单位。例如，在图7-5中，街道的结构在世界坐标空中定义，此坐标系的轴向的一个单位代表10码，而机器人则定义在一个以厘米为度量的完全独立的世界坐标中。虽然每个根结构在其自己的世界坐标系中定义，但每一个显示设备只有一个NPC空间，由所有被提交的结构共享，因为这个坐标空间是显示设备的一个抽象。

7.3.5 通过窗口管理共享屏幕的图像应用

20世纪70年代早期，当设计第一个图形软件包时，在给定的时间内只能运行一个图形应用软件，并且使用整个屏幕。PHIGS的设计开始于20世纪70年代后期，当时这种独占屏幕的操作模式仍占主流，并且窗口管理程序还没有被广泛使用。因此，NPC空间的单位立方体就习惯地被映射到整个屏幕。 255~256

现代的拥有多任务操作系统的图形工作站允许同时运行多个图形应用程序，在窗口管理程序的控制下，共享工作站的资源、屏幕和输入设备。在这种环境中，每个应用程序被分配了一个自己的窗口，这个窗口就像一个“虚拟的屏幕”一样工作。用户通过调用窗口管理器的函数，能够改变窗口大小或移动窗口位置。其主要的优点在于，每个应用程序就像控制整个窗口一样工作，而不必关注其显示屏幕只是实际显示设备的一部分。因而在窗口管理器环境下，一个SPHIGS的应用程序不需要做其他的调整，软件包和窗口管理器合作将一个NPC空间映射到所分配的窗口而不是整个屏幕。

图7-6显示两个SPHIGS应用和一个终端仿真程序同时运行在一个图形工作站上。由于SPHIGS将NPC空间映射到适合窗口管理器所提供的窗口的最大正方形区域，非正方形窗口的一些部分对SPHIGS应用是不能用的，如图示的SPHIGS窗口所表现的桌椅的场景。

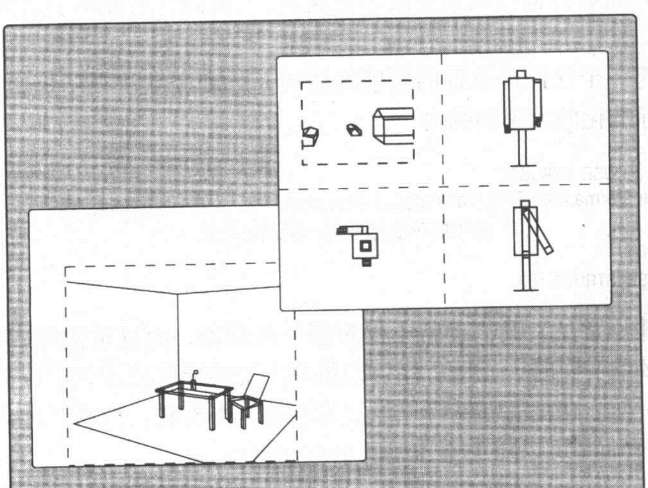


图7-6 两个SPHIGS应用程序，每个应用在其自己的窗口管理器的窗口中运行。虚线表示视口边界

7.4 模型变换

7.3.2节包含了一段C程序，此程序创建一个简单的结构用于房屋的建模。出于简化考虑，我
257 们把房屋的一个角放在坐标原点，房屋的边与主轴对齐，并设置整坐标单位的大小。我们把定义在坐标原点的并与主轴对齐的对象称为是*标准化的*。标准化的对象不仅较具有任意空间位置的对象易于定义（确定其顶点坐标），也易于进行几何操作，如改变大小、方向、位置等。

如果我们要将房屋放在不靠近原点的不同的位置，我们当然可以自己重新计算房屋的顶点，然后用7.3.2节所示的相同的C代码重新创建一个房屋的结构，只是改变了顶点的坐标。但是，我们现在介绍一种强大的变换技巧实现对标准化部件对象的位置和大小的改变。

如同我们在第5章所看到的，我们可以通过将以列向量$[x\ y\ z\ 1]^T$表示的每个顶点乘一个4×4的齐次坐标变换矩阵，来变换一个多边形这样的图元。以下的实现函数生成一个这样的矩阵：

```
matrix_4x4 SPH_scale( double scaleX, double scaleY, double scaleZ );
matrix_4x4 SPH_rotateX( double angle );
matrix_4x4 SPH_rotateY( double angle );
matrix_4x4 SPH_rotateZ( double angle );
matrix_4x4 SPH_translate( double deltaX, double deltaY, double deltaZ );
```

对不同的轴可以有不同的缩放因子，这样一个对象可以被非均匀地被拉伸和扭曲。对于旋转，用度表示的角度参数代表其绕设定的主轴按逆时针运动的角度，主轴的方向沿坐标轴的正无穷指向原点。

矩阵可以用于生成一个结构中的变换元素。下面是元素生成过程：

```
void SPH_setLocalTransformation( matrix_4x4 matrix, mode REPLACE /
    PRECONCATENATE / POSTCONCATENATE);
```

“Local”这个前缀词表示SPHIGS如何显示一个结构。当SPHIGS遍历一个结构时，它存储一个*局部矩阵*，作为已经遍历过的结构的状态信息的一部分。局部矩阵通常都被默认地被初始化为单位矩阵。每当碰到一个setLocalTransformation元素，局部矩阵以某种方式修改：它可以被由mode参数定义的多重操作改变或者代替。在遍历过程中每碰到一个图元，其顶点都将由局部变换矩阵进行变换，然后再进行观察变换以便显示。（我们在后面将会看到，层次结构使这个过程变得更复杂。）

下面的代码创建一个在任意位置的房屋的结构，并利用默认视图提交该结构以用于显示。房屋维持它原来的标准化的大小和朝向。

258
```
SPH_openStructure (HOUSE_STRUCT);
   SPH_setLocalTransformation (SPH_translate(...), REPLACE);
   SPH_polyhedron (...);          /* 这里的顶点像前面一样被标准化 */
SPH_closeStructure();
SPH_postRoot (HOUSE_STRUCT, 0);
```

像这样简单的变换并不常见。我们通常不仅希望平移对象，而且希望能影响它的大小和朝向。当要求对图元进行多重变换时，应用程序可以将每个变换矩阵依次相乘以得到一个局部的合成变换矩阵。通常，一个标准化的部件可以经过先缩放，再旋转，然后平移到达指定位置。就像在第5章中所述，这种顺序可以避免不需要的平移和剪切。

下面的代码创建并提交一个房屋的结构，它从原点移开并被旋转到我们可以看到其侧面而非其前面的位置：

```
SPH_openStructure (MOVED_HOUSE_STRUCT);
```

```
    SPH_setLocalTransformation (SPH_rotate(...), REPLACE);
    SPH_setLocalTransformation (SPH_translate(...), PRECONCATENATE);
    SPH_polyhedron (...);        /* 这里的顶点像前面一样被标准化 */
SPH_closeStructure();
SPH_postRoot (MOVED_HOUSE_STRUCT, 0);
```

对平移矩阵使用PRECONCATENATE模式保证左乘用于合成平移矩阵和旋转矩阵，从而先转动再平移。左乘要远比右乘普遍，因为它和单个变换元素的顺序相对应。因为SPHIGS相对于主轴进行旋转和缩放，所以编程人员必须生成一个将任意轴映射到主轴所需的变换矩阵，然后做变换，然后再向回映射正如第5章所讨论的。

SPHIGS在遍历时间完成变换元素的合成，这样，每次屏幕刷新时，都执行变换单元合成的过程。另一种通过定义连续的变换序列的方法则提高了显示遍历过程的效率，即我们自己在一定义时间合成这些变换，并生成一个变换元素。下面的函数在定义时间内执行矩阵乘法：

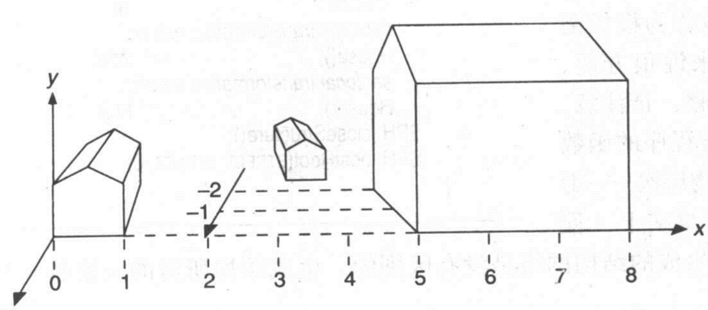
```
matrix_4x4 SPH_composeMatrix( matrix_4x4 mat1, matrix_4x4 mat2 );
```

在前面代码中的两个setLocalTransformation可以用以下代码代替：

```
SPH_setLocalTransformation (
    SPH_composeMatrix(SPH_translate(...), SPH_rotate(...)), REPLACE);
```

这种方法的缺点在于不能对图元的大小和方位进行动态的改变，因为这种改变需要对一系列setLocalTransformation元素的成员做选择性的编辑；另外，整个合成矩阵必须重新计算和指定。 259
提高效率的经验规则是：除非其中的某个变换发生了选择性的更新而不得不单独指定外，在其余的情况下，我们就在定义时使用合成。

让我们来看一下由三所简单房子（如图7-4所示）组成的街道结构。在如图7-7a所示的透视图中，左边是一所民房，右边是一幢大厦，中间是一个小村舍。我们画出平行于x坐标轴的虚线并在x轴上做了标记，指出这些房子的相对位置，并且我们使用SPHIGS的显示模式，表示出多面体线框边，并进行了消隐（见7.8节）。最左边的房子是没有经过任何变换的标准化的房，而另外两所房子则是大小、方向和位置不同的副本。



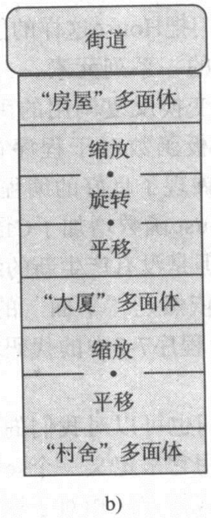


图7-7 a) 构建一个具有三所房屋的街道模型，透视图，b) 街道-房屋模型的结构

如图7-7b的示意图所示，生成这个街道结构的直接的方法就是先定义三个标准化房子多面体，然后进行适当的变换。我们把顺序进行的一组变换元素看成一个单元；其中的第一个元素采用了REPLACE模式，其余均采用PRECONCATENATION方式，相邻的变换之间用乘号（·）隔开。生成整个街道结构的代码如程序7-3所示。

程序7-3 生成图7-7a的代码

```
SPH_openStructure(STREET_STRUCT);
   /* 使用其标准方式定义第一所房屋 */
   SPH_polyhedron(...);

   /* 大厦就是将房屋在x, y, z方向上分别放大2倍、3倍和1倍，绕y轴旋转90°，
      然后平移；请注意它的左面现在是朝前的，并处于(x, y)平面中 */
   SPH_setLocalTransformation(SPH_scale(2.0, 3.0, 1.0), REPLACE);
   SPH_setLocalTransformation(SPH_rotateY(90.0), PRECONCATENATE);
   SPH_setLocalTransformation(SPH_translate(8.0, 0.0, 0.0), PRECONCATENATE);
260   SPH_polyhedron(...);
   /* 村舍就是将房屋均匀缩放0.75倍，不旋转，设置在z轴的负方向、x轴的正方向 */
   SPH_setLocalTransformation(SPH_scale(0.75, 0.75, 0.75), REPLACE);
   SPH_setLocalTransformation(SPH_translate(3.5, 0.0, -2.5), PRECONCATENATE);
   SPH_polyhedron(...);
SPH_closeStructure();
SPH_postRoot(STREET_STRUCT, 0);
```

如程序7-4所示，我们可以定义了一个C函数来生成标准化房子多面体，以消除该标准房子多面体定义中的冗余。因为本例中的房子是由一个简单的多面体调用定义的，这种方法的效果并不明显；如果房子的构造更复杂一点，包含多个图元及属性的定义，这种方法就能明显地减少代码量。除此之外，该方法还有利于程序的模块化：当房子的形状或式样改变时，只需编辑House函数，而不必编辑构造街道结构的代码。

程序7-4 使用一个模板程序对街道进行建模

```
void House()
{
   SPH_polyhedron(...);
}

main()                          /* 主线 */
{
   SPH_openStructure(STREET_STRUCT);
      House();                  /* 第一所房子 */
      set local transformation matrix;
      House();                  /* 大厦 */
      set local transformation matrix;
      House();                  /* 村舍 */
   SPH_closeStructure();
   SPH_postRoot(STREET_STRUCT, 0);
}
```

我们把House这样的能生成定义结构化部件的一系列元素，并能被经过任意的几何变换反复调用的函数称为**模板函数**。模板函数对于程序员来说很方便，并且也展现了良好的编程风格。请注意，虽然House函数增加了C语言程序的函数层次，但是没有产生新的结构层次——街道模型依然是“平面”的。事实上，程序7-4与程序7-3中的代码所生成的结构网络是没有区别的。生成结构所需的元素数也基本相同。

我们也可以对我们的模板函数做的一个改变是让它接受一个变换矩阵作为参数，然后模板函数用它来定义一个setLocalTransformation元素。虽然在某些情况下传递变换参数是方便的，但这种方法相对于原来的方法而言，丧失了其可以在调用模板前定义任意数量变换的通
261 用性。

7.5 层次式结构网络

7.5.1 两层层次结构

到目前为止，我们已经讨论了三种类型的结构元素：输出图元、外观属性和几何变换。下面，我们将介绍SPHIGS系统是如何从结构层次中获得强大功能，以及这种分层结构是如何实现的。这种结构层次是通过使一个元素在运行时遍历调用子结构而实现的，不要把它同前面介绍的模板函数层次结构相混淆。模板函数的结构层次是在对CSS进行编辑时（定义时间）形成的，它产生的是嵌入式元素，而非子结构调用。与此相反，子结构调用引起的结构层次是在显示时对CSS进行遍历而形成的。

调用子结构的**结构执行元素**以如下方式创建：

```
void SPH_executeStructure( int structureID );
```

让我们用一个在CSS中生成一所房屋的函数来替换前面的模板函数（如程序7-5所示）。该函数在主函数中只调用了一次，并且HOUSE_STRUCT从未提交过；它是作为街道结构的一个子对象被调用并显示出来的。注意，这里STREET_STRUCT定义的惟一区别就在于增加了对建造房屋结构的函数的调用，以及把每一个对模板函数的调用都替换为对executeStructure的调用。虽然显示图像跟图7-7a是一模一样的，但结构网络不一样，如图7-8所示，图中每个executeStructure元素都引出了一个箭头。

程序7-5　利用一个从属结构对街道进行建模

```
void  BuildStandardizedHouse()
{
   SPH_openStructure(HOUSE_STRUCT);
   SPH_polyhedron(...);
   SPH_closeStructure;
}

main()
{                                                    /* 主线 */
   BuildStandardizedHouse();
   SPH_openStructure(STREET_STRUCT);
      SPH_executeStructure(HOUSE_STRUCT);            /* 第一所房子 */
      设置局部变换矩阵;
      SPH_executeStructure(HOUSE_STRUCT);            /* 大厦 */
      设置局部变换矩阵;
      SPH_executeStructure(HOUSE_STRUCT);            /* 村舍 */
   SPH_closeStructure();
   SPH_postRoot(STREET_STRUCT);
}
```

提交STREET_STRUCT使得SPHIGS系统通过遍历STREET_STRUCT的结构网络来更新屏幕；遍历采用了深度优先的方式，正如函数/子程序层次结构被执行一样。在前述的例子中，遍历过程先把街道结构的局部矩阵规格化为单位矩阵，接着对房屋子结构做第一次调用，调用时将街道结构的局部矩阵应用到房子的每个顶点，就好像构成房屋的多面体本身是街道结构的图元一样。第一次调用返回后，局部矩阵被置为想要得到的变换的合成矩阵，然后做第二次调用，并将这个新的合成矩阵应用到房屋

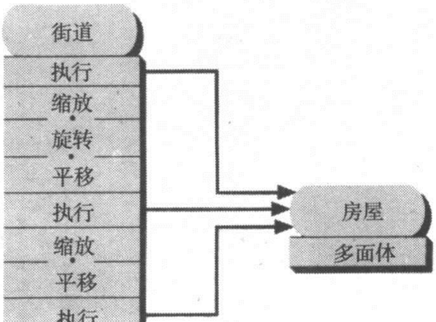


图7-8　表明从属结构的调用的结构网络

顶点，以生成房子的第二个实例。第二次调用返回后，局部矩阵被再次改变；新的合成矩阵再次被应用于房子的顶点以得到第三个房子实例。

我们把一个结构看成是一个独立的实体，其中的图元都定义在它自己的浮点造型坐标系统（MCS）中；这样考虑是为了便于建立低层的标准化的部件。正如我们在5.9节所指出的那样，变换将某个坐标系统下的顶点映射到另一个坐标系统中；这里，SPHIGS采用结构*S*的局部矩阵把子结构中的图元变换到*S*本身的MCS中。

7.5.2 简单的三层层次结构

262~263 作为一个三层层次结构的简单例子，我们对街道例子中的房屋进行了扩展。新的房子由原来的标准化的房子（改称为SIMPLE_HOUSE_STRUCT）和一个烟囱组成，烟囱经过了适当的缩放和平移并直立在房子的右上角。有两种方法可以修改房子的结构，一是把组成烟囱的面直接加在原来的多面体上，二是再在结构中增加一个多面体。我们的选择是把房子分解为两个子体从而引入三层层次结构。这种模块化的好处在于我们可以在烟囱本身的MCS中以标准化的方式（位于原点，单位尺寸）来定义烟囱（如图7-9a所示），然后再经过缩放和平移将其放到房子的MCS中并位于屋顶。要是我们把烟囱直接定义到屋顶并映射到房子的MCS而不进行缩放，很明显我们就不得不对其顶点进行繁杂的计算。然而采用模块化处理，我们只需定义一个标准化的烟囱并使它底面的坡度与屋顶相同；只要满足这个条件，就能进行均匀的缩放和任意的平移。

修改后的房屋结构由以下代码生成：

```
SPH_openStructure (HOUSE_STRUCT);
   SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
   set local matrix to scale/translate standardized chimney onto roof of simple house;
   SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure();
```

当我们用街道结构以变换来实例化这个两层的房子，生成如图7-9b所示的三层层次结构会

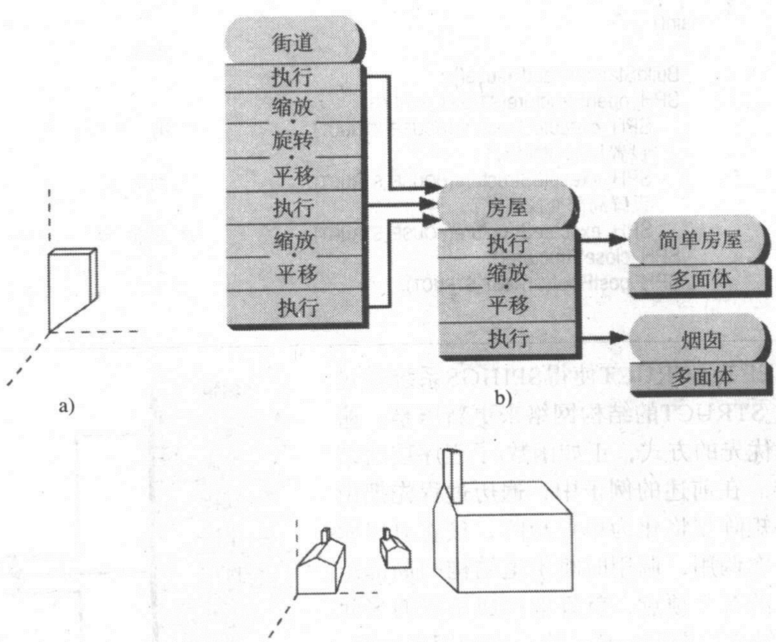


图7-9 a) 三层的街道层次结构的标准化烟囱，b) 三层的街道层次结构的结构网络，c) 从使用三层层次结构得到的结果图像

怎样呢？由于SPHIGS是通过变换其组成元素和子结构来变换一个父结构的，因此我们可以确保这两个图元构件(简单房子和烟囱)是作为一个整体而得到变换的（如图7-9c所示）。这里的关键之处在于街道结构的定义完全不用改变。因此，街道结构的设计者不必关心内部的细节，比如房子是怎样构成的等等，对于设计者来说，这是一个黑匣。

7.5.3 自底向上构造机器人

作为一个更加有趣的典型例子，我们来看一下简单机器人，它采用结构层次造型，并通过不断修改变换以达到动态效果。对于复杂的物体或系统层次结构，通常采用自顶向下的方式来进行构思和描述。比如，一幢计算机系的大楼由各个楼层组成，每个楼层都包括一些办公室和实验室，而每个办公室或实验室都有大量的器具和设备，依此类推。让我们回想一下，这个机器人由上身和下部底座组成；上身包括躯干、头和两支一样的手臂，其中每支手臂又由一个固定的手和一个可以滑动（平移）的手指（作为钳子）组成。

虽然我们设计时采用了自顶向下的方式，但实现时却采用了自底向上的方式，定义一些构造块用于逐层生成高层的构造块的定义，从而形成部件的层次结构。因此，在构造机器人时，我们就定义手指构造块来定义手臂，然后把两个手臂部件的实例拼在上身上，依此类推，部件层次的示意图如图7-2所示，上身的详细结构网络图如图7-10所示。

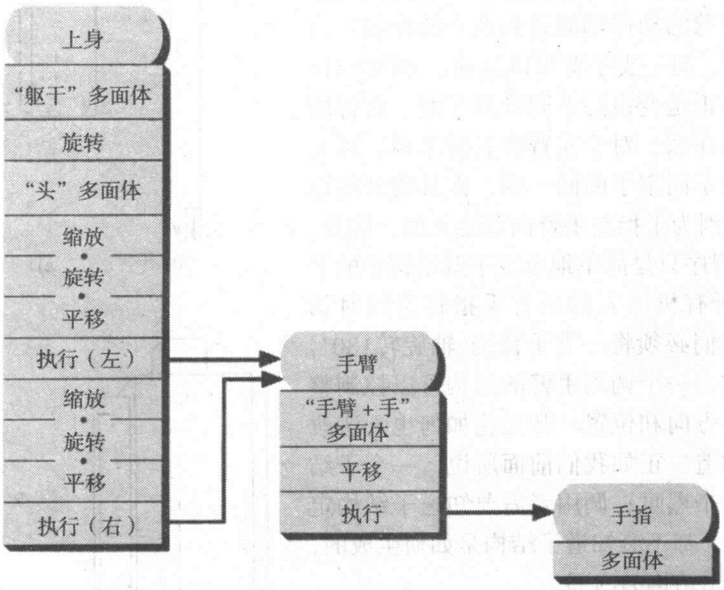


图7-10　机器人上身的结构层次

让我们来仔细分析一下这个自底向上的构造过程所包括的几何结构和变换。用相同的计量单位来定义手臂和手指使得二者之间更容易组合。我们把手指结构定义在它自身的MCS中的标准化位置，在MCS中它沿y轴“悬垂”（如图7-11a所示）。定义手臂结构时采用了与手指相同的计量单位，它由一支手臂＋手的多面体（标准化的，沿y轴悬垂，如图7-11b所示）和一个经过平移的手指结构的调用组成。手指调用之前的平移元素负责将手指从标准化的原点位置平移到手腕上的适当位置（图7-11c）。

手臂调用手指网络是一个两层结构，与房屋–街道的例子很相似。通过编辑手臂结构的平

移元素，我们可以沿着手腕滑动手指（如图7-11d）[⊖]。

接下来，我们构造上身。由于我们想要头部可以转动，故而先定义躯干多面体，然后旋转，再定义头部多面体（图7-11e）。下一步是让这个上身结构调用手臂结构两次。在调用前应当做什么样的变换呢？如果我们主要关心的是使手臂准确定位于上身的MCS中，我们将得到如图7-11f所示的结果：手臂和上身的设计明显不合比例。这个问题很容易解决：我们可以在此之前先进行一个缩放变换（如图7-11g所示）。但是，如果我们要使手臂能够绕连接双肩的轴线转动（就像是行进中的士兵的动作一样），光靠一个缩放变换和一个平移变换是不够的；为达此目的，平移前我们在结构中增加了一个旋转元素。到此为止，我们完成了对整个上身结构的定义；习题7.1就是装配整个机器人。图7-11h展示了左臂旋转元素不为0时的情况。

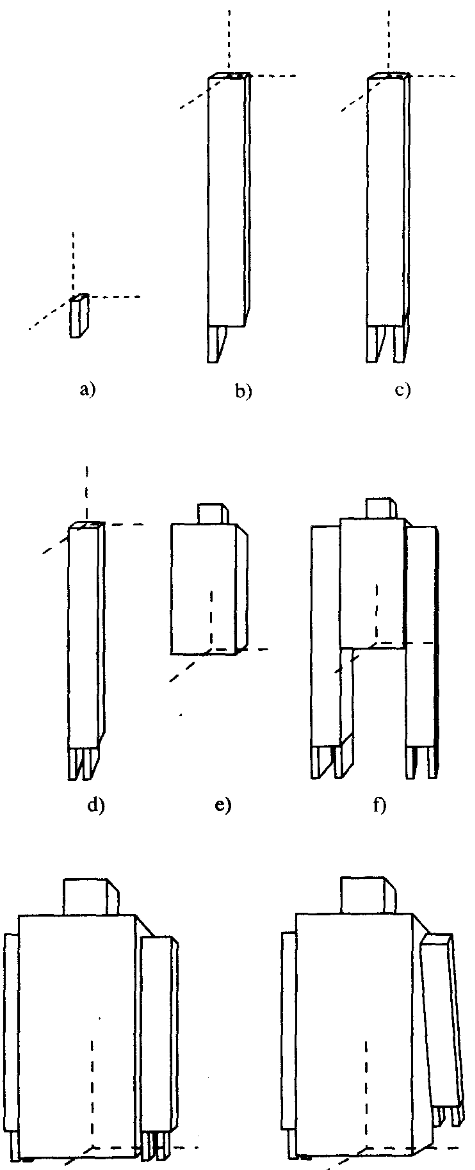


图7-11　构造一个机器人的上身。a)手指，b)固定手臂＋手，c)完整的手臂，d)具有平移的手指的完整的手臂，e)躯体和头，f)具有不合尺寸手臂的上身，g)正确的手臂，h)左手臂旋转

因为每支手臂在调用前都进行了一系列独立变换，因此每支手臂的动作都能得到独立的控制。当一支手臂悬垂时，另一支手臂可以摆动，如图7-11h所示。事实上，正是变换的不同导致了左、右臂的差异。然而，请注意，对于左臂和右臂来说，可滑动的手指不仅位于固定手的同一侧，而且离开的距离也一样，这是因为手指是手臂内部定义的一部分。（实际上，如果程序只是简单地改变手臂结构中的平移元素，那么所有机器人的所有手指将会同时动作！）因此，我们必须将一支手臂绕y轴旋转180°，使两支手臂对称。一个调用手臂的结构可以控制整个手臂的尺寸、方向和位置，却无论如何也无法改变手臂的内部构造。正如我们前面所说，一个子结构本质上就是一个黑匣；调用者需要知道子结构定义哪些几何结构，却不必知道子结构是如何生成的，也就无法影响到子结构的内部。

总而言之，在定义任何结构时，我们只需处理如下问题：该结构所包括的图元和低层子结构，定位该结构的组成部件时所需用到的造型变换，以及影响子结构外观所需用到的属性。我们不必也不能改变低层子结构的内部构造。此外，我们在设计构件时不必关心它们将被怎样调用，因为可以用一系

⊖ 谨慎的读者可能会产生疑问：在此模型中，既然它并未真正连在机械手臂上，手指实际上是如何滑动的。事实上，我们的机器人模型的任何部分都未通过所谓的“关节”与其他部分连接。关节的建模和确定它们的操作模式的约束不在此书的讨论范围。

列的变换来得到预期的大小、方向和位置。在实际应用当中，使构件在其自身的MCS中标准化是很有用的，这样它可以很方便进行缩放和绕主轴的旋转变换。

还有两点需要补充说明一下。首先，一个程序员不必用单纯的自顶向下的方法来设计，或是用单纯的自底向上的方法来实现；与编程技术中使用的“桩模块和驱动模块”类似，可以在结构层次中使用虚子结构。一个虚子结构可以是一个空的结构；事实上，在SPHIGS中允许一个结构调用一个还未生成的子结构，在这种情况下，SPHIGS会自动生成这个子结构并将它初始化为空。在某些情况下，需要使虚子结构是一个与复杂实体大小近似的简单实体（比如，一个平行六面体），复杂实体要稍后才能确定。这种方法使得在所有部件都得以确定前，对复杂的结构层次进行自顶向下的调试成为可能。其次，我们还没有对结构层次和模板函数层次做出比较。到目前为止，我们可以说，当我们需要使用一个部件的多个拷贝，并对其中每个拷贝的外观和布局（而不是它们的内部）进行控制（在一定程度上）时，采用层次结构还是很有效的。我们将在[FOLE90]的7.15节中对模板函数层次和结构层次之间的权衡与折衷给予讨论。

7.5.4 交互式造型程序

交互式三维造型程序通过采用前面所述的自底向上的装配过程，使得物体的层次构造更为容易。大部分的这种程序都提供了一个由许多图标组成的面板，这些图标就代表了程序中的基本部件集。如果制图程序是专用的，那么部件集也是专用的；否则，它们就是诸如折线、 264~268
多边形、立方体、平行六面体、圆柱体、球体之类的通用元素。用户可以选择一个图标以得到对应部件的一个实例，然后对该实例定义一些变换。这些定义是通过输入设备来做的，如鼠标、控制盘等等，并能让用户对实例的大小、方向和位置进行实验，直到满意为止。由于要对三维景物的二维投影中的空间关系进行判断是很难的，更为完善的交互技术是应用三维网格、数值比例、各式各样的滑块以及关于顶点位置的数值反馈信息等等（见8.2.5节）。一些构造程序还允许用户通过组合基本图形部件生成高层部件，并加入部件面板中，以用来组建更高层的对象。

7.6 显示遍历中的矩阵合成

到目前为止，我们已经介绍了程序员是怎样通过自顶向下的设计和自底向上的实现来构造一个模型的。不管模型是怎样构造的，SPHIGS对DAG可采用自顶向下、深度优先搜索把模型显示出来，该DAG的根是已提交的结构。在遍历的过程中，SPHIGS对各层变换和调用中指定的所有几何结构都要进行处理。让我们来看一个自顶向下遍历的例子：根部结构*A*调用结构*B*，*B*又调用结构*C*。这就是说，*C*中的图元从*C*自身的MCS变换到*B*的MCS，以完成对*B*的构造；同样，通过把*B*中的图元（包括调用*C*得到的图元）变换到*A*的MCS中，又完成了对*A*的构造。结果是*C*中的图元变换了两次，先是从MCS_C到MCS_B，再从MCS_B到MCS_A。

采用5.9节中介绍的表示法，以$M_{B\leftarrow C}$表示结构*B*时的局部矩阵，在调用*C*时将MCS_C中的顶点映射到MCS_B中的相应的变换位置。因此，我们把一个顶点从MCS_C映射到MCS_B记为$V^{(B)} = M_{B\leftarrow C} \cdot V^{(C)}$（这里$V^{(H)}$指代表一个其坐标在坐标系*H*中表示的顶点的向量）；类似地，有$V^{(A)} = M_{A\leftarrow B} \cdot V^{(B)}$。因此，为了模拟自底向上的构造过程，遍历程序必须连续应用一系列的变换，以把顶点从*C*映射到*B*，再从*B*映射到*A*：

$$V^{(A)} = M_{A\leftarrow B} \cdot V^{(B)} = M_{A\leftarrow B} \cdot (M_{B\leftarrow C} \cdot V^{(C)}) \tag{7-1}$$

根据矩阵的结合律，$V^{(A)} = (M_{A\leftarrow B} \cdot M_{B\leftarrow C}) \cdot V^{(C)}$。因此，遍历程序只需把两个局部矩阵组合起

来，并将结果矩阵应用于C中的每个顶点。

采用树形表示法，树根位于第1层，后续的子树分别位于第2, 3, 4, …层。根据归纳法可知，
269 对于位于第 j 层(j>4)的任意结构，我们可以通过下式，把处于该结构本身的MCS中的顶点$V^{(j)}$变换为处于根部坐标系统中的顶点$V^{(1)}$：

$$V^{(1)} = (M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} \cdots \cdot M_{(j-1)\leftarrow j}) \cdot V^{(j)} \tag{7-2}$$

由于SPHIGS允许图元在自己的局部MCS中通过局部矩阵做变换，将局部矩阵作用于图元的坐标值就得到了顶点$V^{(j)}$：

$$V^{(j)} = M^{(j)} \cdot V^{(prim)} \tag{7-3}$$

我们用$M^{(j)}$来表示这个局部矩阵，当遍历到这个结构时，$M^{(j)}$用来对该结构的图元变换到它自己的第j层MCS中。若这个结构还调用从属子结构，局部矩阵的作用又有了变化；它将用来把被调用的子结构从第$j+1$层的MCS变换到第j层中去，我们用$M_{j\leftarrow(j+1)}$来表示。这并不意味着矩阵的值发生了变化，而是它的作用发生了变化。

使用结合律合并式(7-2)和式(7-3)，得

$$V^{(1)} = (M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} \cdots \cdot M_{(j-1)\leftarrow j} \cdot M^{(j)}) \cdot V^{(prim)} \tag{7-4}$$

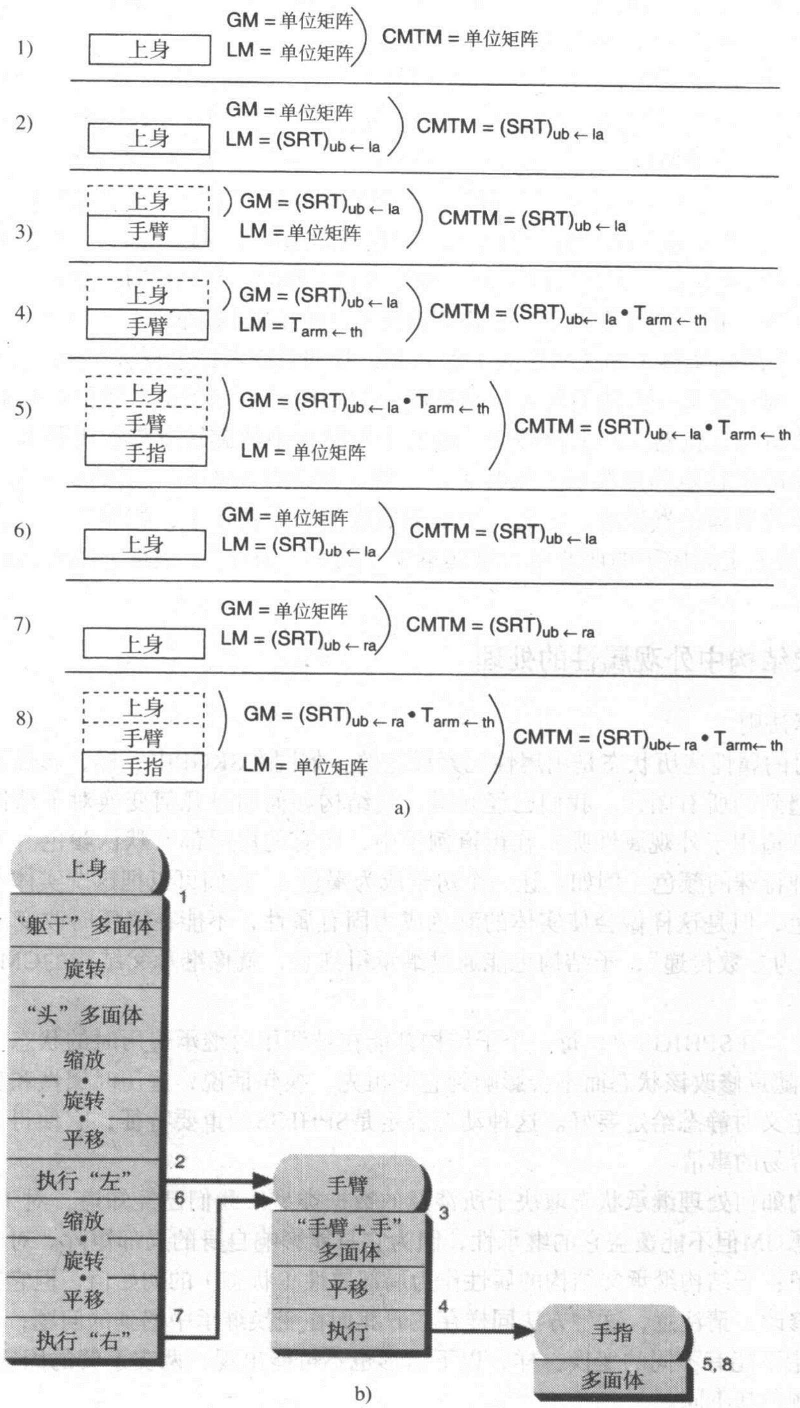
因此，要把第j层的图元变换到根部的MCS(即世界坐标空间)中去，我们所要做的是要自顶向下，把从根部到图元本身所在的结构之间的所有结构的局部矩阵组合起来。这些局部矩阵的合成（见式(7-4)括号中的项）称为**合成造型变换矩阵**（CMTM）。当遍历到第j层结构中的图元时，CMTM是j个矩阵的合成。只有最后一个矩阵（$M^{(j)}$）才能被结构改变，因为一个结构只能修改它本身的局部矩阵。因此，当一个结构被激活时，CMTM中的前$j-1$个矩阵是固定的。当遍历到第j层结构时，前面$j-1$个矩阵的合成称为一个**全局矩阵**（GM）（见式(7-2)括号中的项）。在遍历一个结构期间，保留这个GM对SPHIGS来说是很方便的；当一个setLocalTransformation元素改变了局部矩阵（LM）时，SPHIGS通过把这个新的局部矩阵右连接到GM中，就可以很容易地计算出新的CMTM。

现在我们来总结一下遍历算法。SPHIGS做一个深度优先遍历，调用任何结构前先保存CMTM、GM和LM；然后将子结构的GM和CMTM初始化为继承得到的CMTM，LM初始化为单位矩阵。CMTM用于做顶点变换，并随LM的改变而改变。最后，当遍历过程返回时，恢复父结构的CMTM、GM和LM，继续执行。由于对矩阵做了保存和恢复，父结构对子结构会产生影响，反之则不会。

让我们看一下SPHIGS遍历如图7-10所示的上身-手臂-手指三层层次结构时的情况。我们
270 提交UPPER_BODY结构作为根。图7-12a展示了遍历状态的快照序列；每个快照与图7-12b的结构网络图中标有相应序号的点相对应。

遍历状态通过如图7-12a所示的堆栈维护，这个堆栈是往下增长的，当前的活动结构位于实线矩形框中，其祖先位于虚线框中。与当前活动结构相对应的三个状态矩阵示于堆栈的右侧。弧线描出了变换的作用域：GM所对应的弧线包括了对GM有贡献的祖先结构，CMTM对应的弧线则指明它是GM与LM的乘积。再提醒一下，对于每一组变换，第一个变换矩阵采用REPLACE模式，而其余的都采用PRECONCATENATE模式。这样就保证了结构中的第一个旋转变换只作用于头部，而不会影响其他（如左臂），因为这个旋转变换矩阵将被作用于左臂的第一个缩放变换矩阵所替代。

在图7-12b中标出的点1，表示遍历过程刚执行到根部的第一个元素。由于根节点没有祖先，它的GM是单位矩阵；LM也是单位矩阵，对于任何结构刚开始执行时均是如此。对于点2，LM



271

图7-12 a) 三层层次结构中的遍历状态栈一个时刻的状态，b) 三层层次结构的带注释的结构网络

被置为三个变换（缩放、旋转、平移，简写为SRT）的合成。因此，CMTM也相应的变为单位矩阵GM与合成矩阵SRT的乘积，以用来将手臂子结构变换到上身的MCS中去，于是通过$(SRT)_{ub\leftarrow la}$生成了左臂。接下来，对于点3，遍历过程将执行手臂结构的第一个元素。在期望第2层实例化时，手臂执行的GM在调用点正是其父结构的LM。

在点4，手臂的LM用于定位手臂上的手指($T_{arm\leftarrow th}$)，CMTM更新为GM与LM的乘积。第2层的CMTM成为第3层手指实例化的GM（点5）。由于手指的LM是单位矩阵，于是其CMTM变换的结果就是先从手指的坐标系变换到手臂的坐标系，再从手臂的坐标系变换到上身的坐标系。在点6的位置，遍历过程已经从手指和手臂的调用返回到了上身。上身结构的矩阵又变回到跟调用前一样，因为对从属结构的调用无法改变它的局部矩阵。到了点7，上身的LM又被新的右手臂的合成矩阵所代替。当我们逐层向下遍历到右手臂的手指结构时（点8），CMTM几乎跟点
272 5处的一样；惟一的区别在于用于将手臂定位到上身的第2层矩阵。

要使一个组合实体（如使机器人）动起来，我们只需考虑怎样去影响父结构中的子结构，并针对每个构件定义一些适当的可以动态改变的变换元素。于是，我们可以通过改变整体结构中的旋转矩阵让机器人的上身转动，改变上身结构中的旋转矩阵让机器人抬起手臂，改变手臂结构中的平移矩阵使机器人张开手指。整个层次结构中每一层的变换都是独立进行的，但产生的净效果却是累积的。定义一个动画最难的部分就在于：如何从一个想要得到的结果（比如“机器人走到屋子的西北角，拿起桌子上的一个木块”）倒推，以得到满足这个要求的变换序列。

7.7 层次结构中外观属性的处理

7.7.1 继承法则

遍历时的属性遍历状态是由属性元素设定的，如同在SRGP中一样，属性遍历状态被应用于遍历时遇到的所有图元。我们已经知道，父结构如何通过几何变换对子结构施加影响。什么样的规则适用于外观属性呢？在街道例子中，所有的房屋都有默认颜色。为了给某一个实体结构一种特殊的颜色（例如，让一个房子成为褐色），我们可以把这个实体本身一开始就定义成该颜色，但是这样做会使实体的颜色成为固有属性，不能在遍历时改变。更好的办法是“把颜色作为参数传递”，子结构也能通过继承得到它，就像继承父结构的CMTM作为自己的GM一样。

事实上，在SPHIGS中，每一个子结构都能在被调用时继承遍历时的状态（只要该状态存在），并能随意修改该状态而不会影响到它的祖先。换句话说，遍历时属性和变换是动态绑定的，这比定义时静态给定要好。这种动态绑定是SPHIGS的重要特征，它使得自定义子结构实例成为很容易的事情。

子结构如何处理继承状态取决于所涉及的数据类型。我们已经知道，对于几何变换来说，子结构继承GM但不能覆盖它的继承性，因为它只能影响自身的局部矩阵。对于属性来说，情形更为简单：子结构继承父结构的属性作为局部属性（状态）的初始值，但它能随后对其局部状态进行修改。请注意，这种方法同样存在着我们在变换继承中遇到的问题；正如机器人两支手臂的手指不能有不同的变换一样，以下情形也不可能出现：两支手臂的固定部分颜色相同，而手指的颜色却不同。

在图7-13a所示的结构网络图中，街道结构对房屋子结构设置了颜色。其结果图像见图7-13b，
273 相应的代码在程序7-6中列出。

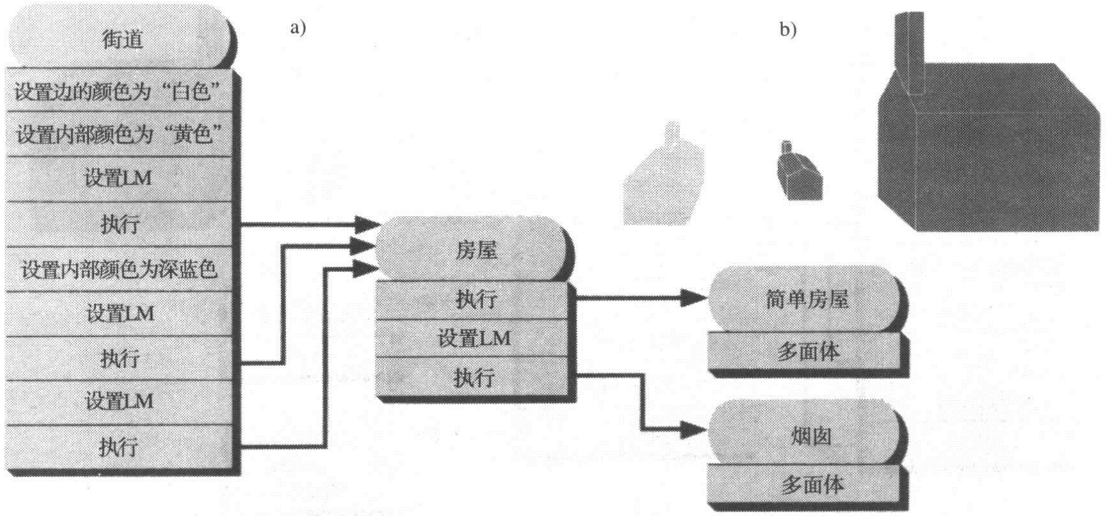


图7-13 在用彩色房屋构建街道模型中使用属性继承。a)结构网络，b)结果图像。（内部颜色由图案模拟。）

子结构中属性可以重置以覆盖继承所得的值。下面的代码段定义了一个修改后的房屋结构，它的烟囱总是红色的。

```
SPH_openStructure (HOUSE_STRUCT);
   SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
   SPH_setInteriorColor (COLOR_RED);
   set up transformations;
   SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure();
```

让我们用这个新的房屋结构与程序7-6中的代码生成的街道结构连接起来。图7-14显示了结构网络以及结果图像。遍历过程从STREET_ STRUCT开始；棱边的颜色属性和内部的颜色属性都被设为默认值。棱边设置为白色，这个值在遍历过程中自始至终保持不变。第一个setInteriorColor函数将HOUSE_STRUCT的第一个实例置为黄色，这样该实例将黄色传递给SIMPLE_HOUSE_ STRUCT，SIMPLE_HOUSE_STRUCT的多面体显示为黄色。当遍历过程从SIMPLE_HOUSE_ STRUCT返回到HOUSE_STRUCT时，内部颜色属性立即被下一个元素变为红色。于是CHIMNEY_STRUCT的引入导致烟囱显示为红面白边。当然，这些操作都不会影响到STREET_ STRUCT的属性组；当遍历过程从HOUSE_STRUCT返回时，STREET_STRUCT的内部颜色属性又恢复为黄色。接着，该内部颜色属性又设为深蓝色，从而为画后两个蓝色房子做好准备。

程序7-6 生成图7-13的代码

```
SPH_openStructure (STREET_STRUCT);
   SPH_setEdgeColor (COLOR_WHITE);

   SPH_setInteriorColor (COLOR_YELLOW);
   起动变换;
   SPH_executeStructure (HOUSE_STRUCT);

   SPH_setInteriorColor(COLOR_NAVY);
   起动变换;
   SPH_executeStructure (HOUSE_STRUCT);

   起动变换;
   SPH_executeStructure (HOUSE_STRUCT);
SPH_closeStructure();
```

7.7.2 SPHIGS的属性及文字不受变换影响

在PHIGS的真正实现中，文字会像其他的图元那样随着变换而变化。因此，对于一辆卡车侧面上的文字，它的透视效果会随着透视缩小效应旋转或显示，就好像字母是由一系列单独的折线和填

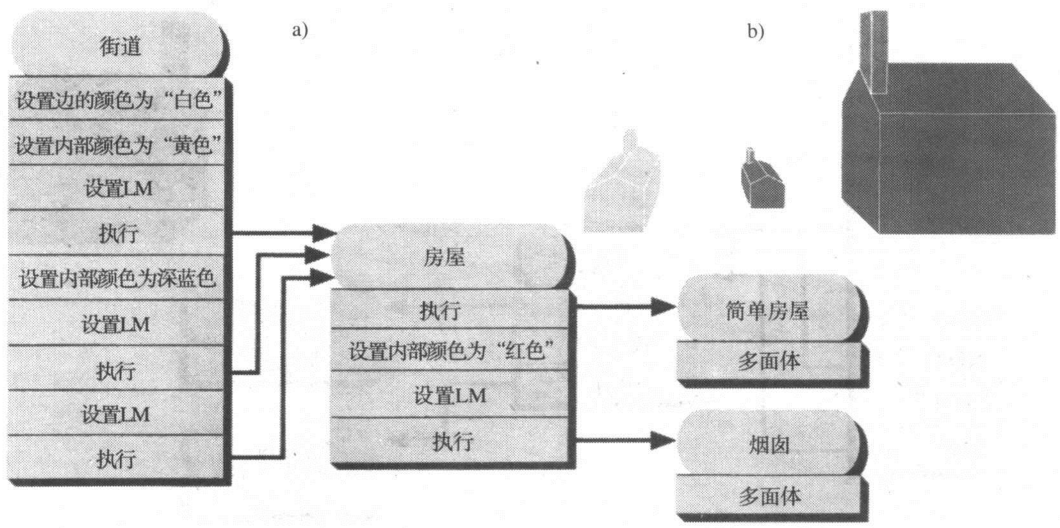


图7-14 改变继承属性的从属结构。a)结构网络，b)结果视图

274~275 充区组成的。同样，虚线中的短划线也要受几何变换和透视缩小效应的影响。然而，由于性能上的考虑，SPHIGS中的属性（包括文字）是非几何的。正如在SRGP中，屏幕上的文字尺寸只取决于字体，甚至连字符串都无法旋转——屏幕上的字符串永远都是竖直的。因此，旋转和缩放变换只影响文字显示的起点，不能影响它的尺寸和方向。SPHIGS中的文字图元主要用于标注。

7.8 屏幕的更新和绘制模式

SPHIGS经常更新屏幕图像以适应当前CSS和视图表的状态。以下动作能使屏幕图像无效：

- 视图表表项的改变。
- 结构被关闭（经过打开和编辑后）。
- 结构被删除。
- 结构被提交或不被提交。

一旦调用SPHIGS完成以上动作中的任意一个，它都必须重新生成屏幕图像以显示所有提交的网络的当前状态。至于SPHIGS如何生成图像则与程序所选择的绘制模式有关。这些模式表示了在再生质量与速度之间做出选择：质量越高，则绘制图像所需的时间就越长。设置绘制模式由下面的过程进行：

```
void SPH_setRenderingMode( int viewIndex, int WIREFRAME / FLAT / LIT_FLAT /
    GOURAUD );
```

这里我们列出了SPHIGS中的四种绘制模式；我们将在第12~14章对它们进行更加充分的讨论。

1. 线框绘制模式

WIREFRAME（线框）模式是速度最快但也是最不逼真的显示形式。这时显示出来的只是由物体的边构成的线条图。此时，视图空间内所有物体的所有边的可见部分都被显示出来，且不做消隐。所有的图元按照时间的顺序（即遍历程序在数据的提交结构网络中遇到它们的顺序）一一画出；这个顺序与7.3.4节中提到的由视图索引确定的显示优先级有关。

在这种模式下，各边的属性将会以其被设计的模式影响到屏幕的外观；事实上，当一个边标志被设为EDGE_INVISIBLE时，整个的填充区域和多面体都将是不可见的。

2. 明暗处理的绘制模式

在其他三种绘制模式下，通过用填充多边形绘制，SPHIGS以更加逼真的方式显示填充区域和多面体。阴影区域的出现严重地增加了绘图过程的复杂度，因为空间顺序变得很重要——物体被遮挡的部分（由于它们被邻近的物体所遮蔽）不应被画出。确定可见面的方法（即消隐） 276
将在第13章中讨论。

对于这三种绘制模式，SPHIGS对各个面可见部分的内部像素生成明暗效果；绘制的质量因模式的不同而不同。对FLAT模式，即本章的插图中经常采用的，一个多面体的各个面都用当前的内部颜色画出，不考虑场景中光源的影响。如同线框方式一样，边的可见部分被显示出来（如果边的属性标为EDGE_VISIBLE）。如果内部颜色被置为背景色，则只有边被显示出来——FLAT方式的这种用法将会得到图7-7a和图7-9c中那样的图形，即消隐后的线条图。

另外两种高质量的绘制模式生成由光源照射得到的图像⊖；光照和明暗处理模型将在第14章中讨论。这些图像的明暗度是非均匀的，每个像素的颜色基于但并不等于内部颜色属性的取值。在LIT_FLAT方式下，同一平面上的各个像素具有相同的颜色，该颜色取决于光线与平面之间的夹角。由于每个小平面都有一个的统一颜色，整个图形看起来棱角分明，相邻平面在公共边上的反差很明显。而GOURAUD明暗处理则消除了这些棱角，使图形显得更为平滑。

在FLAT方式下，边标志属性应被设为EDGE_VISIBLE，因为没有可见边，观察者只能确定出物体的轮廓线边界。但在最后两种高质量模式中，边的可见性通常是关闭的，因为明暗处理能帮助用户确定出物体的形状。

7.9 用于动态效果的结构网络编辑

对于任何数据库，我们都希望不仅能创建和查询它用于显示，还能以一种便利的方式修改它，SPHIGS中的结构数据库也是如此。一个应用程序通过本节所介绍的函数来对结构进行编辑；如果应用程序本身含有自己的模型，那么它就必须保证二者在修改时的前后次序。**运动动力学**需要对模型变换和观察变换进行修改；**更新动力学**的变化要求对结构进行变动乃至替换。当变动相对比较小时，程序员可以选择对结构的内部元素进行修改；然而当变动很大时，通常情况下就只有删除并重建整个结构了。

在本节的剩余部分，我们将提供编辑内部结构的方法；查阅SPHIGS的参考手册能得到有关编辑操作的信息，以及有关函数的详细描述。 277

7.9.1 利用索引和标记访问元素

SPHIGS和PHIGS的基本编辑功能类似于采用行号来定位的老式的行程序编辑器。结构中的元素从1到N建立索引；一旦有元素被加入或删除，同一结构内索引号排在后面的所有元素便相应地把自己的索引号加1或减1。惟一的**当前元素**是其索引被存储在**元素指针**状态变量中的元素。当一个结构随着SPH_openStructure调用而打开时，元素指针置为N（指向最后一个元素）或置为0（结构为空）。该指针在有新元素插入到当前元素之后时加1，在前元素被删除时减1。该指针也能由编程人员通过绝对或相对的定位命令来明确地设定：

⊖ PHIGS PLUS提供的多种控制绘制的功能，包括多光源的位置和颜色的定义，决定物体光照性质的材质属性等，见第12~14章。

```
void SPH_setElementPointer( int index );
void SPH_offsetElementPointer( int delta );
  /* +向前，-向后 */
```

由于在同一父结构中，一个元素的索引会随着它前面元素的添加或删除而改变，利用元素索引号来定位元素指针很容易出错。因此，SPHIGS允许应用程序在一个结构中放置界标元素，称为**标记**。一个标记元素在生成时都给定了一个整型标识符：

```
void SPH_label( int id );
```

应用程序可以通过下面的函数来移动元素指针：

```
void SPH_moveElementPointerToLabel( int id );
```

然后该指针向前移动搜索给定的标记。如果在找到这个标记之前到达结构的末尾，搜索不能成功终止；因此，程序将建议你在搜索标记前把指针移到结构的头部（索引为0）。

7.9.2 内部结构的编辑操作

最常见的编辑操作是在一个结构中插入新元素。一旦元素生成函数被调用，新元素就会立刻放置在当前元素的后面，同时元素指针加1以指向这个新元素。

元素可以通过下列函数删除：

```
void SPH_deleteElement();
void SPH_deleteElementsInRange( int firstIndex, int secondIndex );
void SPH_deleteElementsBetweenLabels( int firstLabel, int secondLabel );
```

278 在以上各种情况下，删除完成后，元素指针将指向被删除元素的前一个元素，所有剩下的元素都将重新编号。第一个函数删除当前的元素。第二个函数删除两个指定元素之间（包括这两个元素在内）的元素。第三个函数有点儿类似，但并不删除两个标记元素本身。

请注意，这些编辑工具都影响整个的元素或元素集；这里并没有提供对一个元素内部的数据域进行选择性编辑的方法。因此，举例来说，当程序员需要改变多面体内部的一个顶点时，就必须重新定义整个多面体。

一个编辑的例子

让我们来看一下对一个简单街道示例进行的修改。现在我们的街道只包括第一个房子和村舍，前者固定，后者可以移动。我们在村舍前建立一个标记，这样我们稍后可以通过修改变换来移动村舍。

为了移动村舍，我们在此重新打开街道结构，将指针移到标记，再定位到变换元素，替换变换元素，然后关闭结构。关闭结构后屏幕将自动更新，于是村舍在新位置上显示出来。有关代码见程序7-7，操作的顺序见图7-15。

程序7-7　编辑图7-15中的街道结构的代码

```
SPH_openStructure(STREET_STRUCT);
   /* 当一个结构被打开时，元素指针初始化其最表端。
      我们必须移动指针到开始处，于是我们可以搜索标记。 */
   SPH_setElementPointer(0);
   SPH_moveElementPointerToLabel(COTTAGE_TRANSLATION_LABEL);
   SPH_offsetElementPointer(1);      /* 现在指针指向变换元素 */
   SPH_deleteElement();              /* 在此通过删除/插入组合来替代 */
   SPH_setLocalTransformation(newTranslationMatrix, PRECONCATENATE);
SPH_closeStructure();
```

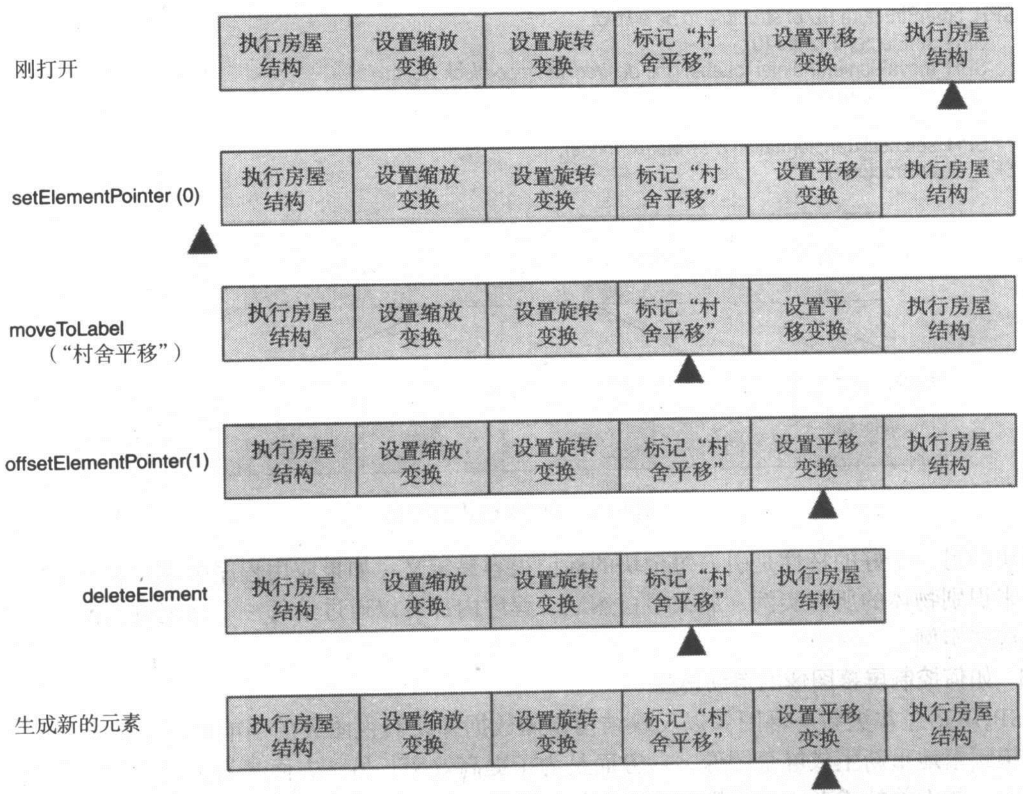


图7-15 编辑过程中的结构序列。黑色三角形表征元素指针的位置。（为了图示的目的已简写了调用的语法。）

7.9.3 改进编辑方法的一些实例块

前面的编辑例子建议我们在需要编辑的元素前放置标记，但是要创建如此多的标记显然是一件麻烦事。下面介绍几种避免这个麻烦的办法。首先是用两个标记把要修改的一组元素包括起来，然后利用标记来删除或覆盖整个元素组。另一种常用的方法是把元素分组处理为固定的格式，每个组引入一个标记。要编辑组内的任一元素，只需把元素指针移到该组的标记，然后通过偏移量将指针从该标记定位到组的内部。由于组的格式固定，偏移量可以很容易地通过一个整数确定。

该方法的一个特例就是，通过在结构执行（structure-execution）元素前附加一个属性设置元素的通用列表，来设计实例化子结构的一个标准方法。这种元素序列的一个典型格式被称为**实例块**（instance block），如图7-16所示；首先用一个惟一的标记来标识这整个块，然后是一个内部颜色设置，接着是三个基本变换，最后是符号结构的调用。

我们可以创建一些符号常量来提供偏移量：

```
#define INTERIOR_COLOR_OFFSET   1
#define SCALE_OFFSET            2
#define ROTATION_OFFSET         3
#define TRANSLATION_OFFSET      4
```

实例块的固定格式的使用保证了对于任何实例都能用同样的方式来修改某个特定的属性。为了改变一个特定实例的旋转变换，我们采用如下代码：

```
SPH_openStructure (ID of structure to be edited);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (the desired instance-block label);
    SPH_offsetElementPointer (ROTATION_OFFSET);
    SPH_deleteElement();
    SPH_setLocalTransformation (newMatrix, mode);
SPH_closeStructure();
```

279~280

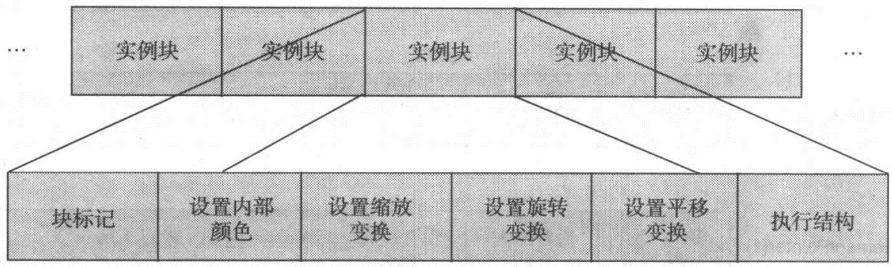


图7-16 实例块格式的样品

实例块的另一个好的特性是引入每个块的标记很容易定义：如果应用程序中维护着一个内部数据库来识别物体的所有实例，那么很自然地，程序内部可以通过给每一个标记设置惟一的值来识别这些实例。

7.9.4 如何控制屏幕图像的自动再生

SPHIGS经常更新屏幕图像以反映结构存储数据库及其视图表的当前状态。但是有时我们不希望频繁地重新生成屏幕图像，一方面是为了提高效率，另一方面是为了避免用户看到编辑过程中一些无关的或者比较模糊的连续变化的图像。

其实我们可以看到，SPHIGS在编辑结构的时候是将重新生成图像的功能挂起，这样不管发生了多少变化，图像只有当该结构被关闭的时候才会重新生成。为了提高效率，还可以采用“批处理”更新方法。因为“批处理”中图元的任何一个删除或变换操作都有可能破坏屏幕图像，从而需要进行图像的修复处理甚至重新遍历一个或多个视图中的已提交的网络。但不管怎么样，让 SPHIGS 在再生屏幕之前一次性计算多次操作的累积的效果显然会快于依次计算每次操作之后的图像并刷新。

连续性改变不同结构也有和上面类似的结论，例如，连续调用deleteStructure删除结构及其子结构时。为了避免这个问题，应用程序可以在这些改变之前将屏幕自动再生功能挂起，并在操作之后再解除挂起。挂起和解除挂起的函数为：

```
void SPH_setImplicitRegenerationMode( int ALLOWED / SUPPRESSED );
```

当屏幕再生功能被挂起时，应用程序仍然可以调用

```
void SPH_regenerateScreen();
```

281 显式地要求屏幕再生。

7.10 交互

SRGP 和 SPHIGS 的交互模块是基于 PHIGS 规范的，因此它们设置设备的模式和属性以及获取量度的手段是一样的。SPHIGS 键盘基本上与 SRGP 相同，不同点在于在 NPC 空间上 SRGP 返回的原点不含有 z 坐标值。SPHIGS的定位设备带有一个通常为常量的 z 坐标值的附加域。另外，SPHIGS 增加了两个交互功能。第一个是**关联拾取**，即增加了对用户拾取的物体标

识的定位功能。另一个是支持菜单的选择设备，这在相关的手册中有说明。

7.10.1 定位器

SPHIGS 的定位器返回 NPC 坐标系下的光标位置（其中$z_{NPC}=0$），以及包含该光标且具有最高优先级的视图的索引号。

```
typedef struct {
    point   position;          /* [x, y, 0]NPC屏幕位置 */
    int     viewIndex;         /* 其视口包含光标的最高
                                  优先级视图的索引号 */
    int     buttonOfMostRecentTransition;
    enum {
        UP, DOWN
    } buttonChord[MAX_BUTTON_COUNT];   /* 一般为1~3 */
} locatorMeasure;
```

当两个视口有重迭，且光标含于其交集中，在第二个域中返回在视图表中具有最大索引号的视口。这样，就可以利用这些视图索引建立起视图的输入和输出优先级。视图索引域非常有用的原因很多。考虑一个允许用户交互地指定视口的大小的应用程序。比如可以移动或改变窗口管理器的窗口。响应一个改变窗口的提示，用户只须拾取视口内的任何位置。应用程序可以根据视图索引号确定哪个视图被拾取，而不必通过检测一个点是否在一个矩形域内，对视口边界做逐个判断。对于涉及只用于输出的视图的应用程序，则可以根据返回的视图索引号判断是否需要调用一些相应的函数。

7.10.2 关联拾取

因为 SPHIGS 的程序员是以已建模的实体为单位来考虑问题的，而非组成图像的像素；所 282
以应用程序如果能够确定用户拾取的图像的实体标识将是非常有用的。因此本节将讨论定位器的最基本用途，即为关联拾取过程提供输入的 NPC 点。我们已经知道，在 SRGP 中，关联拾取在平面世界中确定**命中**问题是非常简单的。我们认为其图像与定位的位置足够近的图元被用户选中。当物体是相互重叠时，会有多次命中，一般选择最后画的物体，因为它被认为在最“上面”。这样在二维关联拾取程序中是以时间的逆序来检测图元的，最先被检测到的物体就是被拾取的物体。但是如后面所提到的原因，在三维中拾取对象时，这种重叠的层次结构将变得相当复杂。好在 SPHIGS支持这种应用。

1. 层次结构中的拾取

我们先想一下拾取的层次结构引入的复杂性。第一问题是用关联拾取来确认被拾取的物体时应当返回哪些信息？这个时候一个结构标识（ID）是不够的，因为一个结构标识无法区分一种结构的多个实例。只有**全路径**，即沿从根部到被拾取图元的完整祖先的描述，才能够提供惟一的标识。

第二个问题是当某个特定图元被拾取时，在拾取的层次结构中，用户所选中的应当在第几层？例如，当用户将光标放在机器人的手指处时，用户的本意是去拾取手指、手臂、上身还是整个机器人？有时，用户想拾取的是整体；有时只是想去拾取其中的一部分；任何一个层次结构都是有可能的。有些应用程序通过提供反馈机制允许用户依次沿着层次结构从图元浏览到根部，目的是确切地指定到所需要的层（参见习题 7.8）。

2. 比较的准则

当我们确实需要在三维中比较与已定义的物体何等接近时，由于定位器只能提供二维的 NPC 坐标值，所以与定位点相对应的图元的 z 坐标值是无法被利用的。也就是说，SPHIGS 进行比较时只能根据其屏幕图像的坐标值，而不能利用图元在世界坐标系中的坐标值。但不管怎样，如果

一个物体被命中，则它必定是关联拾取的候选图元。在线框模型中，SPHIGS拾取的是在遍历过程中遇到的第一个候选物。因为线框模型不包含明显的深度信息，所以用户不可能期望深度相关的关联拾取。（这种策略的效果是优化了关联拾取。）在真实感的绘制模型中，SPHIGS 拾取的候选物是最接近视点的**击点**（即NPC点位于与用户点击方向垂直的图元曲面上），即最前面的点。

3. 关联拾取工具

为了完成关联拾取，应用程序调用 SPHIGS 的关联拾取工具，它的参数是一个 NPC 点和一个视图索引号，而这些参数通常是上一次与定位器交互返回的值。

```
void SPH_pickCorrelate( point_3D position, int viewIndex,
    pickInformation *pickInfo);
```
283

返回的信息通过拾取路径标识被拾取的图元及其祖先，拾取路径在程序7-8中以C语言数据类型的格式描述的。

程序7-8 拾取路径存储类型

```
typedef struct {
   int      structureID;
   int      elementIndex;
   enum {
      POLYLINE, POLYHEDRON, EXECUTE_STRUCTURE
   } elementType;
   int      pickID;
} pickPathItem;

typedef pickPathItem pickPath[ MAX_HIERARCHY_LEVEL ];

typedef struct {
   int      pickLevel;
   pickPath path;
} pickInformation;
```

当没有图元接近光标位置时，返回的 *pickLevel* 的值是 0 并且 *path* 域的值是没有定义的。当 *pickLevel* 大于 0 时，*pickLevel* 的值是从根部到被拾取图元的路径的长度，即图元在网络中的深度。而且此时*path*数组的项从 [1] 到 [*pickLevel*] 分别返回路径中从根部到被拾取图元的结构元素的标识。最深层（即项[*pickLevel*]）标识的元素就是被拾取的图元；其他层（从项[*pickLevel* − 1] 到 [1]）标识的元素都是一些结构执行。*path*中的每个标识对应一条记录，该记录给出了包含该元素的结构的结构ID，该元素在其所在的结构中的索引、代表该元素类型的代码以及（下面将要讨论的）该元素的拾取ID。

图7-17用图7-10所示的机器人上身的结构网络，并说明在该结构的显示图像中几个拾取返回的拾取信息。

拾取路径如何惟一标识结构（该结构在层次中可能调用任意多次）的每一个实例？例如，我们如何识别是拾取机器人的左手指还是拾取右手指？如图7-17中的点 a 和 点 e 所示，这两个手指的拾取路径只是在其根部不同。

相对结构ID而言，拾取标识符实际上是一种解决拾取相关的更好方案。虽然元素的索引号能够用于标识每个独立的元素，但是在结构被编辑时它常常会发生变化。所以采用拾取ID更为简单，因为当编辑其他元素时拾取ID不会受到影响。拾取ID的默认值是0，并且被包含在结构中。可以通过调用下面的过程来创建拾取ID元素：

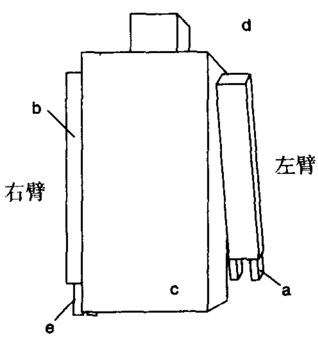
284

```
void SPH_setPickIdentifier( int id );
```

这些拾取ID在显示遍历时被忽略，另外也没有继承的概念。当SPHIGS 开始遍历任何结构时，不管一个结构是子结构还是根，拾取ID的初始值都是0。由于这两方面的原因，拾取 ID 不同于结构的属性。在一个结构内的多个图元可以分别拥有惟一的 ID，也可以共享一个ID，这样就很好地解决了同一个结构内任意的关联拾取，这也正是应用程序所需要的。虽然标记与拾取 ID 采用不同的机制，前者用于编辑，后者用于关联拾取，但它们在应用的过程中常常是相关联的。尤其在结构使用7.9.2节提到的实例块技术组织时，拾取 ID 元素同时也是该实例块的一个组成部分，并且拾取 ID 本身的值通常被设成与块标记相同的整数值。

例7.1

问题：通过提供用户交互来增强机器人动画。房间中给定一定数量的物体，并允许用户（使用定位设备）来选中机器人应该拾取的物体。为简化问题，机器人可以是单臂的。



参见图7-10所示的结构网络

(a) level = 3
path[1] : struct UPPER _BODY, element 7
path[2] : struct ARM, element 3
path[3] : struct THUMB, element 1

(b) level = 2
path[1] : struct UPPER_BODY, element 11
path[2] : struct ARM, element 1

(c) level = 1
path[1] : struct UPPER_BODY, element 1

(d) level = 0

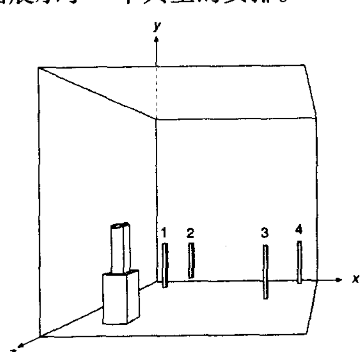
(e) level = 3
path[1] : struct UPPER_BODY, element 11
path[2] : struct ARM, element 3
path[3] : struct THUMB, element 1

图7-17 关联拾取的例子

解答：

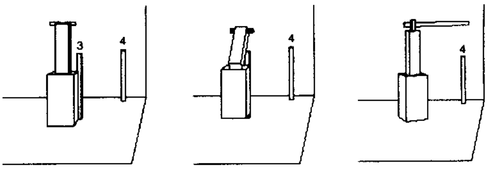
用户视图。用户看到一个简单的房间，在房间的左前方有一支单臂机器人等着，在其后方是一些悬挂的棒。用户可使用鼠标并单击左键来选中任何物体，而机器人将处置它。用户可在 285
任何时候按 q 键退出。

物体的安放。我们安放机器人能够操纵的物体，以便一个简单的策略将满足它接近、抓取和删除任一物体。物体横向（x方向）均匀地安放，并限制在房间的深度内（z方向）。这些物体散播在一个方便的高度上。下图展示了一个典型的安排。



这样安放物体，允许机器人清晰地沿着平行于x轴的路径移动，直到物体的前方与其中的一个对齐为止（x方向对齐）。于是再沿平行于z轴的方向朝着物体移动，其向前移动的距离必须使机器人在易于确定物体的范围内。从该位置，机器人可以抓取物体，举起它，然后转身，并沿原路返回——沿平行于z轴的直线。我们也允许机器人在z轴方向上出了房间（例如到达走

廊)，然后再横向移动（平行于x轴）。注意，对于所有物体按这个方案工作只需要物体的两个坐标（x, z）作为参数。下面的图示序列展示了机器人移近一个选中物体（棒3）、弯下抓取它、准备离开房间去处置已经选中的棒。



286 一旦机器人出了屏幕，用户就无法觉察它完成了什么动作。因此，我们要尽可能简单地去掉机器人所举的物体，并重新定向机器人以便让它返回到屏幕上。为此，我们只要将物体从机器人层次上分离，重置手指的位置，并将机器人移回到它等待的地方即可。

该算法的C代码实现（提供用户交互以控制机器人动画的程序）如下：

```
/* 定义程序中使用的必要常量 */
/* 定义房间、物体和机器人的必要几何量 */

main(argc, argv)
int   argc;
char **argv;
{

/* 设置初始视图、SPHIGS颜色和参数 */

   /* 现在准备事件输入 */
   SPH_setInputMode(KEYBOARD, EVENT);
   SPH_setKeyboardProcessingMode(RAW);

   SPH_setInputMode(LOCATOR, EVENT);
   SPH_setLocatorButtonMask(LEFT_BUTTON_MASK);

   ShowWorld(OUR_ROOM);              /* 显示房间的初始视图 */
   terminate = NO;
   do {
      whichdev = SPH_waitEvent(INDEFINITE);
      switch (whichdev) {
         case KEYBOARD:
         SPH_getKeyboard(keymeasure, KEYMEASURE_SIZE);
         if (keymeasure[0] == 'Q' || keymeasure[0] == 'q')
            terminate = YES;
         break;
         case LOCATOR: {
            SPH_getLocator(&curr_locmeasure);
            /* 如果按钮按下，我们对拾取予以检测…… */
            if (curr_locmeasure.button_chord[LEFT_BUTTON] == DOWN) {
               SPH_pickCorrelate(curr_locmeasure.position,
                                 curr_locmeasure.view_index,
                                 &pick_info);
               if (pick_info.pickLevel > 2) {     /* 物体置于第二级 */
                  if (pick_info.path[1].structureID == OBJECT_SET)
                     MakeAndRunPlan(pick_info.path[1].pickID);
               }
            }
            break;
         }                    /*LOCATOR*/
      }                       /* switch */
287 }                         /* do */
```

```
    while (!terminate);
    SPH_end();
}

void  MakeAndRunPlan(int object_id)
{
    double x, z;

    x = objects_info[object_id].x;
    z = objects_info[object_id].z;

    /* 在x方向移动与物体对齐： */
    MoveRobot(x, Z_WAIT);
    /* 面向物体并靠近它： */
    SpinRobot(NORTH, CLOCKWISE);
    MoveRobot(x, z + DIST_OF_APPROACH);

    /* 取物体…… */
    LowerArm();
    Grab();
    Pickup(object_id);
    RaiseArm();
    /* 取到 */

    /* 去掉物体 */
    SpinRobot(EAST, COUNTERCLOCKWISE);
    SpinRobot(SOUTH, COUNTERCLOCKWISE);
    MoveRobot(x, Z_HALL);
    SpinRobot(EAST, CLOCKWISE);
    MoveRobot(X_OFFSCREEN, Z_HALL);

    /* 放下手臂…… */
    LowerArm();
    UnGrab();
    Letgo();
    RaiseArm();

    /* 并返回到等待区域 */
    SpinRobot(WEST, CLOCKWISE);
    MoveRobot(X_WAIT, Z_HALL);
    SpinRobot(NORTH, COUNTERCLOCKWISE);
    MoveRobot(X_WAIT, Z_WAIT);
    SpinRobot(EAST, COUNTERCLOCKWISE);
}

void Pickup(int id)
/* 从房间里删除相应的棒状物体
   并把它加到机器人的手（臂）上 */
{
    matrix temp_matrix;                /* 变换矩阵 */

    SPH_openStructure(OBJECT_SET);
    SPH_setElementPointer(0);
    SPH_moveElementPointerToLabel(objects_info[id].label);
    SPH_offsetElementPointer(1);
    SPH_deleteElement();
    SPH_executeStructure(NULL_OBJECT);
    SPH_closeStructure();

    SPH_openStructure(OUR_ARM);
    SPH_setElementPointer(0);
```

288

```
    SPH_openStructure(OUR_ARM);
    SPH_setElementPointer(0);
    SPH_moveElementPointerToLabel(OUR_ARM_DRAW_ROD);
    SPH_offsetElementPointer(1);
    SPH_deleteElement();
    SPH_executeStructure(objects_info[id].struct_id);
    SPH_closeStructure();
    /* 展示画面： */
    SPH_regenerateScreen();
}
```

7.11 高级问题

关于SPHIGS和标准PHIGS，有几个方面是不适宜在本书中详细地介绍，我们在此概述其中一些最重要方面。更多细节参见[FOLE90]的第7章。

7.11.1 其他输出特性

1. 属性包

标准的 PHIGS 提供了间接设置属性值的机制。应用程序初始化的过程中可以将属性值存储在它的**属性包**（attribute bundle）中。动态修改对象外观的简单机制是在不改变结构网络的前提下改变属性包表中包的定义。

2. 高亮度与不可见性的名字集

SPHIGS 支持两种传统的反馈机制，在应用程序中它们常常与 SPHIGS 的拾取功能相关联。这两反馈机制分别是加亮物体和使物体不可见。前者的典型应用是在用户拾取物体时提供反馈，后者是使屏幕只显示必要的信息。在 SPHIGS 的参考手册中阐述了如何在遍历的执行过程中往名字集中添加或从名字集中删除名字的结构元素。

3. 图像交换与元文件

虽然 PHIGS 和其他标准的图形软件包通过系统无关或设备无关提高可移植性，但是由于
289 性能的原因，在一个特定环境中实现的软件包常常具有不可移植的高度优化。PHIGS 的归档文件（由图形标准委员会定义）是在给定时间的结构数据库的一个可移植的快照，这样就使得不同的 PHIGS 实现几何模型的共享。PHIGS 实现同时也支持书写元文件，元文件可以包含应用程序当前在显示表面上所显示的内容的快照。

7.11.2 实现问题

为了显示提交的结构网络，SPHIGS 通过递归下降的方法遍历其构件结构元素，并根据元素的类型对该元素执行相应的操作。这种将模型映射为屏幕图像（硬拷贝）的显示过程在 PHIGS 中指的是显示遍历，但更一般的是指**绘制**；相应软硬件中的实现指**绘制流水线**。

1. 遍历

优化再生图像使得尽可能少地遍历 CSS 相当困难，因为一些微小操作的作用也可能是巨大的。例如，当编辑结构时，很难断定屏幕的多少部分需要再生。

2. 绘制

图7-18阐述了实现显示遍历的绘制流水线的概念。第一个步骤是 CSS 本身的实际深度优先遍历。（相反，如果所用的是立即模式的图形软件包，则应用程序可以遍历应用模型或过程性地产生图元和属性。）遍历过程遇到的每个图元都会被传递给流水线的剩余部分：首先，（第5

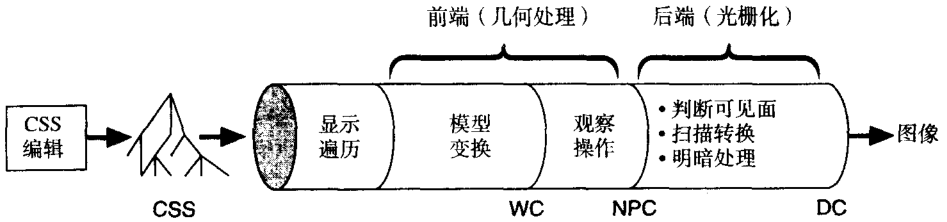


图7-18 SPHIGS的绘制流水线

章所讲的）模型变换被用来将图元从模型坐标系映射到世界坐标系；然后，观察操作被用来变换和裁剪图元使它适合于规范视见体；接着再映射到（第6章所讲的）NPC并行管中。既然这个过程与显示设备无关，并且采用浮点坐标来处理顶点几何结构，那么遍历后面的流水线的部分通常指的是几何处理子系统。 290

流水线的后端取得了经过变换、裁剪的图元并产生了像素，我们将这种像素处理称为*光栅化*。在线框模式下，这个过程显然很直观：（通过缩放和平移，忽略z）NPC 坐标很容易映射成整数设备坐标；然后调用基本的光栅图形软件包中的画线函数进行实际的扫描转换。然而采用经明暗处理的绘制却相当复杂，它由如下的三个子过程组成：**可见面判定**（从视点上看，判定图元的哪个部分是可见的），**扫描转换**（判定图元图像所覆盖的像素），以及**明暗处理**（判定这些像素的颜色）。这三个子过程的实际执行顺序由绘制模式和实现方法决定。从第12章到第14章将详细描述光栅化子过程。

3. 通过范围检测的优化方法

前面所讲述的遍历过程无条件地遍历网络的内容。但是通常并不是所有的网络对象都是可见的，因为进行遍历的时候，有效的模型变换和观察变换有可能会导致网络中的大部分位于视见体外。为了实现这种优化，我们需要一个简单的方法来计算任意复杂物体的边界和一个有效的方法将这些边界与 NPC 视口进行比较。

4. 动画和双缓冲技术

某些简单SPHIGS实现的副作用是：组成动画的“帧”之间，观察者可以看到屏幕正在被删除，并且或多或少可以看到物体是如何随着遍历的过程被重新画到屏幕上的。这种视觉效果可以通过双缓冲来减少：它通过一个屏外画布或位图来保存下一帧，然后再将这一帧复制到屏幕上。

5. 关联拾取

在关联拾取的过程中，SPHIGS 遍历已提交给视口的网络，而那些指定的点就位于这些视口内。另外，这个过程的遍历几乎与显示时的遍历相同，既要维护模型变换矩阵，又要执行大量的绘制流水线。可用不同方法来优化关联拾取遍历，包括解析的及经裁剪的命中检测。解析命中检测的一个例子是函数PtInPolygon，它可用于检测在明暗处理绘制模式下是否落在填充区域及面元上。有一个常用的判断 NPC 光标位置是否在多边形内的算法，它基于2.1.3 节介绍的奇偶校验规则，从光标所在位置发出一条射线然后计算它与多边形相交的次数。该算法遍历边表，检测交点，并处理特殊情况（例如，交点与顶点重合，边与射线共线）。3.5节介绍的多边形扫描转换算法处理的问题非常类似，稍微修改就可以当做函数 PtInPolygon 使用。 291

7.11.3 层次模型的优化显示

1. 省略

我们可以将一个建筑物构建成由各个部分组成的层次结构。例如，它由很多楼层组成，每一楼层由办公室组成，如此等等。层次结构的内部节点不包含任何一个图元，图元只出现在诸如砖头、

平板以及由多面体组成水泥板之类的层次上。虽然这种表示方法可能在建筑上非常有用，但在显示上就不那么有用，例如我们有时只想看看低精度的、去掉模糊的无关的细节之后的简单图像（这也使得绘制更快）。术语省略（elision）指的是由显示遍历程序所做的不再进入子结构的决定。省略可以通过**修剪**、**剔除**以及**层次细节**来实现。

2.参考结构

有些 PHIGS 和 PHIGS PLUS的标准实现允许非标准形式的结构执行，称为**参考结构**。它绕开了代价昂贵的状态保存和恢复的ExecuteStructure（结构执行）机制。一种优化是增加一个ReferStructure操作，使得程序员能够将它用于所调用的子结构并不具有任何属性设置元素的情况。

7.11.4 PHIGS中层次模型的局限性

1. 简单层次结构的局限性

正如1.3节中所提到的那样，有些应用程序并没有给数据提供真正的结构（如离散的绘图数据）或者最多只是对数据进行（部分）排序（例如以代数表示的函数）。许多其他的应用程序更自然地采用网络方式表示，即一般（有向）图（可能带有层次子网）。这其中包括电路和电子线路图、传输和通信网络以及化学工艺图。说明这种简单层次结构在一定的模型下的不足的另一个例子是Rubik 的魔方，这种魔方是一些构件的集合。在任何一种变换后，它的网络和层次（即层、行和列）都会发生根本性的变化。

对于其他类型的模型，这种简单的层次结构也是不足的。例如，绘图仪上的笔架是由水平和竖直的手臂移动的，因此也隶属于这些手臂。简而言之，不管应用程序模型是显示出纯层次结构、不具有层次的网络、具有交叉的网络中的层次结构，还是多重的层次结构，SPHIGS 都可以用来显示该模型；但是我们可能并不希望或不能在最大限度地应用结构的层次。

2. SPHIGS“参数传递”的局限性

结构的黑盒特性对于模块化是有好处的，但正如下面机器人模型的例子那样，它具有局限性。例如，如何构造具有两支相同手臂的机器人，并使机器人用它的右手臂从桌子上拿起一个圆柱，然后再带着圆柱离去？层次结构与一般结构实例的区别在于不同层次级别的变换设置。层次

292 结构不支持结构的实例的原因是结构层次既不具有函数层次的参数传递机制，也不具有控制流构造。

7.11.5 层次建模的其他形式

1. 函数层次

从纯数据层次到纯函数层次的范围中，结构层次几乎总在数据那一端，因为它缺乏控制流。相反，模板函数（即定义模板对象的函数，它由图元或对子模板函数的调用组成）可以使用参数与任意的控制流。

2. 数据层次

不同于函数层次，数据层次非常适于动态创建。与模板函数层次相同的是，它也可以用于立即模式和保留模式图形软件包的连接中。

3. 利用数据库系统

既然通用数据库比专用数据库系统更为强大，我们就应该在计算机图形学中采用标准的数据库系统[WELL76, GARR80]。但是，这样的数据库是用来处理存在辅助存储器中的大量数据，并按人的时间尺度来提供响应时间。

7.11.6 其他（工业）标准

7.1节中我们说明的在PHIGS上下文讨论的许多特点，在它的竞争者（如OpenGL及HOOPS）中也具备。本节，我们简要地介绍这些软件包的不同处，以及PEX与PHIGS的关系。

首先，我们需要区分三维图形包的客户端/服务器（C/S）协议和过程的API（应用程序接

口)，如用于实现应用的PHIGS API之间的差别。API（也就是应用级的子程序库或软件包）使用更低级的C/S协议来实现分布式计算所需要的信息交换。对于选择在C/S环境下工作的二维图形应用，X Window系统已经成为其工业标准有一些时候了。通常，客户端和服务器的进程物理地运行在不同的机器上，它们由局域网（LAN）或广域网（WAN）连接起来。在X Window系统中，服务器进程用来管理显示器，而客户端进程包含了应用代码和图形软件包；两个进程间的通信是通过一种进程间通信协议（IPC）进行的，该协议具有命令序列的形式，每一个命令通常包括了一种操作及其参数。

PEX（原来是PHIGS Extensions to X的缩写，但现在它已不是以PHIGS为中心）是三维分布式图形在X Window系统上的扩展。PEX仅仅有输出功能，其输入由标准X Window系统的输入协议来处理。

PHIGS、HOOPS和PEXlib是在PEX协议之上最常用的API。PEXlib类似于二维图形的X Window系统子程序库Xlib，是PEX协议之上的一个“薄库”，它允许访问全部协议功能，但不支持高度的抽象。PHIGS和HOOPS隐藏了低级功能，但比纯PEXlib调用低效。OpenGL并不运行在PEX之上，而包含了它自己开发的用于分布式图形的协议。

OpenGL只有绘制功能，其通用的API提供了二维和三维功能，包括建模、变换、颜色、光 293
照、平滑明暗处理及其他高级特点，如纹理映射（14.3节）、非均匀有理B样条（NURBS）（9.2节）和运动模糊（12.4节）。它的有些绘制功能比PHIGS PLUS更强，因为它们直接由Silicon Graphics公司的硬件所支持。和PEX一样，OpenGL也支持立即和保留两种图形模式。OpenGL是与操作系统、窗口系统无关的软件包，并已经在UNIX操作系统下与X Window系统集成。

HOOPS与PHIGS有几个重要的不同。因为它是为单一客户而设计实现的，其全部实现是兼容的，而不像PHIGS作为官方标准那种要求允许太多的灵活性，以便适应个别的实现。

HOOPS在字体、图像数据和高级绘制方面比PHIGS提供更完善的支持。除PHIGS PLUS提供的绘制功能外，HOOPS还支持几种形式的全局光照模型，包括光线跟踪、辐射度和一种光线跟踪和辐射度组合的绘制模型（见第14章）。

和PHIGS相似，HOOPS也支持结构层次，但有一个主要差别。PHIGS的结构中包含了图形元素和属性的顺序依赖表。编辑这些表时要求程序员具有如何及在何处去设置、修改属性的详细知识。相反，HOOPS图段中的所有图元具有相同的属性状态，它被提取并放到图段头内。这样简化了程序员的任务，可生产更多模块化的与顺序无关的模型，并通过简化、簿记和删除在绘制流水线上切换的内容，允许一个更高性能的实现。

小结

本章介绍了几何模型，重点是表示零件装配的层次模型。虽然许多数据与对象类型并不是层次的，但至少许多人造的对象或多或少的具有这种特性。PHIGS及其可替换的软件包（如OpenGL、HOOPS及PEXlib)，以可观的复杂度为代价提供了高水平的功能。它们提供了采用多边形、多面体、曲线和曲面的层次化几何造型机制，并能像X Window系统那样与窗口管理器共存工作、提供客户端/服务器的计算功能。

PHIGS 的子集 SPHIGS 提供几何对象，特别是以多边形和多面体的层次模型保存的几何对象的有效的自然的表示方法。因为这些软件包保存对象的内部数据库，程序员很容易就可以对数据库做出修改，并且软件包自动地产生更新后的视图。这样应用程序就可以建立和编辑数据库，特别是响应用户的输入，而软件包则负责产生数据库的特定的视图。这些视图应用了大量

的绘制技术，对图像质量与速度进行了权衡与折衷。软件包也提供了定位器和选择输入设备以及关联拾取，使得对象可以在层次结构的任何一个层次上进行拾取。高亮度过滤器和可见性过
294 滤器被用来启用或禁用针对物体外观的其他形式的控制。

由于结构特性及其查找和编辑手段的限制，这样一种专用系统最适合于运动动力学和光更新动力学，尤其是如果结构数据库能够在优化为 PHIGS 外围设备的显示终端上维护。如果许多结构数据库在连续图像之间必须更新，或者如果应用程序数据库能够快速地被遍历且计算机和显示子系统之间不存在瓶颈，那么在立即模式下使用图形软件包且不保留信息会更加有效。

结构层次位于纯数据层次和纯函数层次之间。它具有数据层次的特点——拥有动态编辑的优点。同时通过属性遍历状态机制，它还允许将参数传递给（几何和外观属性）子结构的简单形式。但是，由于缺乏一般控制流的构造，这种参数传递机制是有限制的，而且结构无法有选择地在子结构不同实例中设置不同的属性。相反，模板函数可以被用来建立(层次)结构的多个副本，它们在结构上是相同的而在子结构的几何属性或外观属性上是不同的。另外，它们可以用来驱动立即模式软件包。

SPHIGS 是面向由多边形与多面体构成的几何模型，特别是具有层次特性的几何模型。第9章和第10章介绍的几何模型有更加复杂的图元以及图元的组合。在进入更加先进的造型方法之前，我们在后面几章中先介绍一下交互工具、技术和用户界面。

习题

7.1 a. 通过给图7-11中的机器人添加一个基座，使得它的上身可以在这个基座上转动来完成该机器人模型，并建立起它在房间内运动的简单动画。

b. 建立一个产生动画的 SPHIGS 应用程序。在该动画里，有一个只有一支手臂的机器人，它走到一张放着一个物体的桌子那里，拿起物体，并带着该物体离开。

7.2 增强机器人的动画，使得动画的三视图同步地显示，这其中包括俯视图和“机器人的视野”视图（即机器人在走动所“看到”的视图）。

7.3 更新递归显示遍历程序，使得它维护为每个结构保存的 MC 的范围信息。假定在编辑完结构S之后，S结构记录中的extentObsolete布尔域被设置。同时也假定给定任何一个图元，函数返回的该图元的NPC 范围总是有效的。

7.4 给定线段的 NPC 的端点坐标和定位器测量值，设计分析计算候选线段的命中点的算法。

7.5 设计分析计算候选填充区域的命中点的算法。

295 7.6 采用伪代码，设计只支持线框模式的递归关联拾取的遍历过程。

7.7 完成用在关联拾取中的函数 PtInPolygon。处理当射线通过顶点或者射线与边重合时的特殊情况。参见 [PREP85] 和 [FORR85] 中所讨论的本问题的微妙之处。

7.8 设计拾取用户界面，使得用户可以指定想要的层次结构的层。针对机器人模型通过写一个使用户可以高亮机器人的各个部分（从独立的部分到整个子系统）的应用程序来完成并检
296 测你设计的界面。

第8章 输入设备、交互技术与交互任务

由于现在硬件和软件的价格已经低得足以给办公室和家庭提供高效的计算能力，因此高质量的用户界面已成为向各类用户提供计算能力的“最后一个有待开发地带”。正如近来软件工程由软件结构发展为软件活动（软件过程）一样，用户界面工程这一新的领域也正在形成用户界面原理和设计方法学。

用户界面的质量常常决定了一个用户是喜爱还是轻视一个系统，系统的设计者受到赞扬还是指责，系统在市场上是成功还是失败。一个交互式图形应用的设计者必须对用户的需求十分敏感，即用户需要容易学习且功能强大的界面。

桌面隐喻用户界面（具有窗口、图标和下拉菜单，而且它们都充分利用了光栅图形）由于易于学习并且几乎不需任何打字技巧而备受欢迎。这些系统的大多数用户都不是计算机编程人员，与许多编程人员喜好大相径庭，他们不喜欢老式的、不易学习的、面向键盘的命令语言界面。设计、测试及实现用户界面的过程是很复杂的，指导原则及方法参见[FOLE90; SHNE86; MAYH90]。

本章讨论的重点是输入设备、交互技术和交互任务。这些都是构造用户界面的基本部件。输入设备是用户向计算机系统输入信息时使用的硬件部分。在第4章中我们已经讨论过很多这类设备，本章介绍其他设备，并讨论选用某种设备而不用另一种的原因。8.1.6节描述面向三维 297
交互的输入设备。我们继续使用定位设备、键盘设备、选择设备、定值设备和拾取设备等逻辑设备类型，这些类型是由SRGP、SPHIGS和其他与设备无关的图形软件包中所采用的。我们也要讨论用户界面的基本要素：交互技术以及交互任务。交互技术是指使用输入设备向计算机中输入信息的方法，而交互任务对采用交互技术输入的信息按基本类型分类。交互技术是设计精巧用户界面的基本部件。

交互任务是用户送到一个信息单元的输入项。四种基本的交互任务分别是**定位**、**文本输入**、**选择**和**定量**。定位交互任务中的信息输入单位当然是位置。与之类似，文本输入任务产生一个文本字符串，选择任务产生一个对象标识，定量任务产生一个数值。许多不同的**交互技术**可用于一项给定的交互任务，例如，一项选择任务可以通过使用鼠标从菜单中选择菜单项来实现，也可以使用键盘输入选择名、按功能键或者使用语音识别设备来实现。类似地，单个设备也可以用于多项不同任务，如鼠标经常用来定位和选择。

交互任务不同于前面几章讨论的逻辑输入设备。交互任务由“用户需要完成什么”来定义，而逻辑输入设备是按任务*如何*由应用程序和图形软件包来实现进行分类。交互任务是以用户为中心的，而逻辑输入设备是一个针对编程人员和图形软件包的概念。

本章中的很多论点在以下文献进行了更加深入的讨论，请参阅Baecker和Buxton[BAEC87]，Hutchins、Hollan和Norman[HUTC86]，Mayhew[MAYH90]，Norman[NORM88]，Rubenstein和Hersh[RUBE84]，Shneiderman[SHNE86]所著的教科书和[FOLE90]，Salvendy的参考书[SALV87]以及Foley、Wallace和Chan的综述[FOLE84]。

8.1 交互硬件

我们在这一节介绍4.5节中没有提及的一些交互设备，详细阐述这些设备是如何工作的，

并且对各种设备的优缺点加以讨论。本节按4.5节的逻辑设备分类进行讨论，它可作为4.5节内容的更详细延伸。

各种交互设备的优缺点可以分三级讨论：设备级、任务级和对话级（即几种交互任务序列）。**设备级**以硬件本身的特点为中心，不涉及由软件控制的设备使用方面。例如，在设备级，我们会注意到一种形状的鼠标手握起来可能比另一种形状的更加舒服，数据输入板比游戏杆占用更多的空间。

298 在**任务级**，我们可以针对同一项任务使用不同设备，对交互技术进行比较。从而可能得出以下结论：有经验的用户通过功能键或键盘输入命令比通过菜单选择更快；或者用户使用鼠标来拾取可见的对象比使用游戏杆或者光标控制键更快。

在**对话级**，我们考虑的不仅是一个个独立的交互任务，而且考虑这些任务组成的序列。手在设备之间移动时需要花费时间：虽然用鼠标完成定位任务通常比用光标控制键更快，但是如果用户的双手已经位于键盘之上，并且需要继续在光标定位以后按顺序用键盘完成下一个任务，这时用光标控制键定位可能比鼠标更快。

本节所讨论的设备级，重点关注的是设备的**足迹**（footprint, 指其占用的工作区域）、操作员疲劳度以及设备分辨率。设备的其他一些重要特性（如费用、可靠性、可维护性等）随着技术的发展而变化太快，这里就不讨论了。

8.1.1 定位设备

按三种相互独立的特性来给定位设备分类非常有用：绝对设备或相对设备、直接设备或间接设备、离散设备或连续设备。

绝对设备（如数据输入板、触摸板）有一个参考区域或原点，并报告相对于原点的目标位置。**相对设备**（如鼠标、跟踪球和控制速度的游戏杆）没有绝对原点，只报告相对于原先位置的目标变化情况。相对设备可用于描述位置的大幅度变化；用户可以沿桌面移动鼠标，向上举起将它放回到最初的开始位置，并再次移动。数据输入板也可编程用做相对设备：指示笔从“远”状态移到“近”状态（即靠近输入板）以后读入第一个(x, y)坐标，用其后读入的所有坐标减去该坐标，仅产生x坐标和y坐标的改变量，这些改变量可加到原先的(x, y)位置。这一过程一直延续到指示笔回复到“远”状态。

相对设备不宜用于数字化绘图，而绝对设备可以。相对设备的优点在于应用程序可以将位于屏幕上任何位置的光标重定位。

用户可以使用**直接设备**（如光笔、触摸屏）用手指或其他代用品直接指点屏幕，而使用**间接设备**（如输入板、鼠标或游戏杆）用户则是用不接触屏幕的设备在屏幕上移动光标。对于后一种情况必须学会眼-手协调的新形式；不过，家庭和游戏厅中计算机游戏的激增已经创造了一种环境，在此环境中，很多不常用计算机的用户已经学会了此类技巧。但是，直接指点会引起手臂疲劳，尤其是对不常用计算机的用户。

连续设备是平滑的手动可以产生平滑的光标移动的设备。输入板、游戏杆和鼠标都是连续设备，而光标控制键则是**离散设备**。连续设备最典型的特点是允许比离散设备更自然、更容易、
299 更快速的光标移动，大多数连续设备比光标控制键更容易在任意方向上移动。

使用连续设备进行光标定位的速度受**控制-显示比率**的影响，控制-显示比率一般称为C/D率[CHAP72]，它是手的移动（控制）与光标移动（显示）之间的比。该比率大有益于精确定位，但这种快速移动过程很枯燥；比率小有益于加快光标移动速度但不利于精确定位。所幸的是，对于一个相对定位设备而言，该比率不必固定不变，而是可以随着控制-移动速度的变化

而灵活改变。快速移动表示用户正在做粗略的手动，所以使用小的比率；而移动速度降低时，C/D率则相应增大。C/D率的改变使得用户可以用鼠标在一个15英寸的屏幕上精确地给光标定位而甚至不必移动一下自己的手腕。对于间接离散设备（光标控制键）而言，也存在相似的技术：每个单位时间光标移动的距离随着按键时间的延长而增加。

在没有支撑点的情况下，如果胳膊伸向屏幕，那么使用直接设备进行精确定位是相当困难的。请试一下用这种姿势在黑板上写你的名字，然后与你正常书写的名字比较。如果将屏幕的放置角度调整到接近水平的位置，可以减轻这个问题。另一方面，间接设备允许手后掌放在支撑点上，这样可以更有效地使用手指的精细运动控制能力。不过，在绘图时并不是所有连续间接设备都同样令人满意的。试一试分别用游戏杆、鼠标、输入板上的指示笔写你的名字。用指示笔写得最快，结果也最好。

8.1.2 键盘设备

众所周知的QWERTY键盘已经伴随我们很多年了，富有讽刺性是这种键盘最初的设计目的是为了减慢打字员的打字速度，这样打字机的印字锤就不会那么容易卡住了。研究表明更新型的Dvořák键盘[DVOR43]比QWERTY键盘更快一些[GREE87]，这种键盘把元音和其他高频字符放在手指最容易击打到的位置，但没有被广泛接受。很多非专业打字用户有时使用按字母顺序组织的键盘，但越来越多的人正在使用QWERTY键盘，多项实验表明QWERTY键盘胜过按顺序组织的键盘[HIRS70；MICH71]。

其他面向键盘的考虑事项不涉及硬件而是涉及软件设计，如让用户不必同时按下Ctrl键或Shift键而输入频繁使用的标点符号或校正字符，将危险的操作（如删除）分配给远离常用键的那些键等。

8.1.3 定值设备

某些定值设备是**有限的**，就像收音机上的音量控制旋钮只能转到防止进一步旋转而设置的 300
截止位置。有限的定值设备输入一绝对的值，而一个连续转动的电位器可以向任一方向旋转**无限次**。设定一个初值，无限的电位器可用于返回一绝对的值，否则，返回值视为相对值。提供某些反馈信息能使用户判定当前确定的是什么样的相对值或绝对值。在定位设备部分讨论的C/D率在使用滑动式电位器和旋转式电位器输入值时也可以使用。

8.1.4 选择设备

功能键是一种常见的选择设备。功能键的位置影响着它们的易用性：固定在阴极射线管(CRT)斜面上的键比键盘上的键或分立装置上的键用起来更困难。脚踏开关可以在用户的双手没有空闲、而又必须经常关闭开关的场合使用。

8.1.5 其他设备

在这一部分内容中，我们讨论一些较为少见、尚处于试验阶段的二维交互设备。语音识别器非常有用，因为它们将用户的双手解放了出来去发挥别的作用。语音识别器将模式识别方法应用于我们说话时产生的波形。典型地，波形被分成一定数量的不同频率的波段，每个波段中波形振幅随时间变化的情况形成模式匹配的基础。可是，模式匹配过程中会产生一些错误，因此尤为重要的是使用识别器的应用程序要提供方便的校正能力。

各种语音识别器在以下方面有所不同：是否识别特定说话人的必须经训练的说话波形；能否识别连续语音，还是个别单词或短语。与说话人无关的识别器只有非常有限的词汇量，一般只包含数字和高达1000个单词。

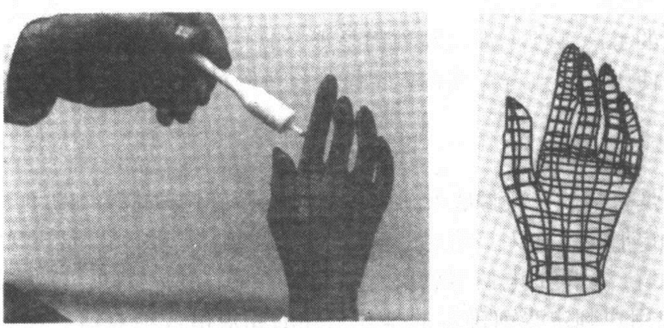
数据输入板已经以多种方式延伸使用。多年以前，Herot和Negroponte使用一种试验性的压

力敏感的指示笔[HERO76]：当压力高、绘画速度慢时表示用户正在仔细画线，这种情况下完全按画出的效果记录曲线；压力低、速度快表示正在快速画线，这种情况下仅记录连接两个端点的直线。近期市场上可以买到的输入板[WACO93]包括这种对压力敏感的指示笔。由输入板产生的3个自由度可在多个方面进行创造性应用。

8.1.6 三维交互设备

一些二维交互设备很容易扩展到三维交互应用中，游戏杆可以有一个手柄作为第三个维数（图4-15），跟踪球可以做成除了能感受到绕两个水平轴的旋转，还可以感受到绕垂直轴的旋转。但是在这两种情况下，利用设备进行手动与在三维空间中做相应运动，这两者之间并没有什么
301 直接联系。

有很多设备可以记录三维的手部运动。例如，Polhemus 3SPACE三维位置和方向传感器采用三根发送天线与三根接收天线之间的电磁耦合。发送器三根天线的线圈相互垂直，形成一个笛卡儿坐标系，并依次产生脉冲。接收器有三根类似安装的接收天线，每次发送线圈发脉冲时，在每个接收线圈中产生电流。电流强度大小不仅依赖于接收设备和发送设备之间的距离，也依赖于发送设备线圈与接收设备线圈之间的相对方向。由三个连续脉冲产生的九个电流值组合，用于计算接收设备的三维位置和方向。图8-1展示了该设备的一项常用用途：数字化三维物体。



a) b)

图8-1 a) 用于数字化三维物体的Polhemus三维位置传感器，b) 线框显示结果。
（3Space数字化仪，由佛蒙特州Colchester的Polhemus公司提供。）

数据手套（DataGlove）记录了手的位置、方向和手指的移动。如图8-2所示，它是带有小而轻的传感器的一个数据手套。每个传感器是一条不长的光纤电缆，其一端有一发光二极管，另一端有一个光敏晶体管。在对弯曲敏感的区域电缆的表面是不平滑的。当光纤折曲时，某些LED的光线丢失，因而光敏晶体管就收到很少光线。另外，手套上的Polhemus位置和方向传感器将记录手的移动。戴了数据手套后，用户能抓取目标，移动和旋转目标，再释放目标。因此数据手套提供了十分自然的三维交互[ZIMM87]。彩图6说明这一概念。

人们做了相当多的努力来创造一种常称之为**人工现实**或**虚拟现实**的环境，这是完全由计算
302 机生成的环境，具有现实的外观、行为和交互技术[FOLE87]。用户配戴一个安装在头部的立体显示器来显示左、右眼视图，头部的Polhemus传感器使头部位置与方向的变化引起立体显示器显示内容的变化，数据手套允许三维交互，麦克风用于发布语音命令。彩图7显示了各种设备的组合情况。

其他几种技术可用于记录三维位置。一种是使用光传感器，发光二极管放在用户身上（要么在一点上，如手指尖，要么遍布全身测量身体运动情况），光传感器安装在用户工作的半明半暗小

房间的角落高处，依次增强发光二极管的亮度。传感器可以确定发光二极管所处的平面，这样发光二极管的位置就在三个平面相交之处。(通常还使用第四个传感器，以防其中一个传感器看不见发光二极管。)指尖及其他点上的小反射器可以代替发光二极管，传感器拾取反射光而不拾取发光二极管发出的光。

Krueger[KRUE83]开发的一种传感器可以记录二维环境中手和手指的运动。用一台电视摄像机录下手的运动情况，利用增强对比度和边缘检测等图像处理技术发现手和手指的轮廓。不同的手指位置被解释为各种命令，用户可以抓取和操纵物体，如彩图8所示，这项技术可以通过使用多个照相机扩展到三维环境中。

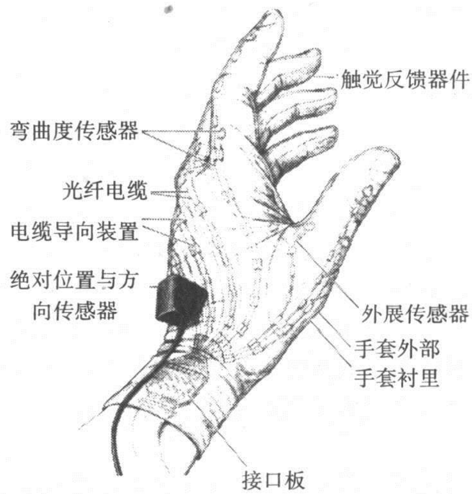


图8-2 VPL数据手套，展示用于传感手指运动的光纤电缆和Polhemus位置与方向传感器。(摘自J.Foley的《高级计算界面》，Scientific American公司1987年出版，版权所有。)

303

8.2 基本交互任务

作为一个基本的交互任务，交互系统的用户输入一个在应用环境中具有一定意义的信息单元。这样一个单元应多大？例如，将一个定位设备移动一小段距离，输入的是一个信息单元吗？如果新位置是针对一些应用目的，诸如重定位对象或指定曲线的结束端点，则输入的是一个信息单元。但如果这个重定位动作仅仅是用户将光标移动到菜单项顶部这一连串重定位动作中的一步，则菜单选择才是一个信息单元。

基本交互任务（BIT）是不可分割的，也就是说，如果将基本交互任务分解成更小的信息单元，那些更小的单元本身对于应用将毫无意义。本节讨论BIT，下一节讨论组合交互任务（CIT）。CIT是此处讨论的基本交互任务的集合，如果将BIT视为原子，那么CIT就是分子。

用于交互式图形学基本交互任务的完整集合包括定位、选择、文本输入以及定量。本节描述各种基本交互任务，并讨论针对每种任务的一些交互技术。不过，交互技术非常之多，不可能一一列出，况且我们也无法预料新技术的发展状况，只能尽可能地讨论各种技术的优缺点。请记住，一项专门的交互技术可能在一些场合很好用，但在另一些场合则不好用。

8.2.1 定位交互任务

定位任务包括给应用程序指定一个(x, y)或(x, y, z)坐标位置，实现这一任务常用的交互技术要么是将屏幕光标移动到希望的位置然后按一下按钮，要么就是在实际的或模拟键盘上键入希望的位置的坐标。定位设备可以是直接设备或间接设备、连续设备或离散设备，也可以是绝对设备或相对设备。此外，可以在键盘上显式地键入移动光标命令，如向上、向左等等，或者可以将相同的命令发到语音识别装置。而且，各种技术可以结合使用——控制光标的鼠标可以用来近似定位，箭头键可以用来将光标移动一个屏幕单位进行精确定位。

有两类定位任务：空间定位任务和语言定位任务。在**空间**定位任务中，用户知道目标位置在什么地方，目标位置在空间上与附近的元素有联系，例如在两个矩形之间画一条直线或者将第三个对象放在两个对象中间。在**语言**定位任务中，用户知道位置(x, y)坐标的数值。对于空间定位任务，用户需要反映屏幕上实际位置的反馈信息；对于后一种任务，需要反馈位置的坐标。 304
如果提供了错误的反馈形式，用户必须在心里将一种形式转换成另一种形式，两种反馈形式都

可以通过既显示光标又显示其坐标值来解决，如图8-3所示。

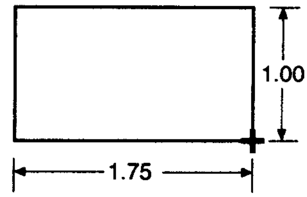


图8-3 关于构造对象尺寸的数字反馈信息。随着光标（+）移动改变对象的高度和宽度，这样用户可以将对象调整到想要的大小

8.2.2 选择交互任务——大小可变的选项集合

选择任务就是从一个**选项集合**中选取一个元素，典型的选项集合包括命令、属性值、对象类和对象实例。例如，一个典型画图程序中的线型菜单是属性值的集合，而这种程序中的对象类型（直线、圆、矩形、文本等等）菜单是对象类的集合。有些交互技术可用于对这四种选项集合中的任一种进行选择，而其他交互技术则通用性较小。例如，不论选项集合是什么类型，点击一个集合元素的可视化表示，就可以选取该元素。但另一方面，虽然功能键非常适用于从命令集合、对象类集合或属性集合中选取元素，但难以在绘图时将一个个独立的功能键分配给每一个对象实例，因为该选项集合的大小可变，而且经常很大（超过可用功能键的数目），同时由于用户创建并删除对象，使对象集合变更非常快。

我们使用术语（*相对*）*固定大小的选项集合*和*大小可变的选项集合*来区分选项集合。前者包括命令、属性、对象类选项集合，后者包括对象实例选项集合。“相对”是指这些选项集合都可能随着新的命令、属性、对象类（如绘图系统中的各种符号）的定义而发生变化，但是集合大小变化并不频繁，而且通常不会变化太大。另一方面，大小可变的选项集合可能变化得相当大，并且变化频繁。

本节讨论专门适用于变化可能性很大的大小可变的选项集合的各种技术，包括通过名字选择与通过定位选择技术。下一节讨论专门适用于（相对）固定大小的选项集合的各种技术，除了复杂应用中大的（但相对固定大小的）命令集合外，这些选项集合一般比较小。将要讨论的技术包括键入或说出名字、缩写词及其他表示集合元素的代码；按下与集合元素相关的功能键等（类似于在键盘上键入一个字符）；在菜单中点击集合元素的可视化表示（文本表示或图形表示）；循环遍历集合直到想要的元素被显示出来；用连续定位设备做特殊运动。

1. 通过名字选择对象

用户可以键入选项的名字。这种想法很简单，但是，如果用户不知道对象的名字怎么办？或者像经常容易发生的那样，同时显示几百个对象怎么办？又或者用户没有理由知道对象名字怎么办？尽管如此，这项技术对多种情况都有用。首先，如果用户极有可能知道各种对象的名
305 字，就像一支舰队的指挥官知道各艘舰船的名字一样，那么用名字引用对象就很明智，并且比定位更快，尤其是当用户可能需要翻滚显示器屏幕查看想要的对象时更明显。其次，如果显示器太拥挤杂乱以致于难以通过定位来选取，以及不能进行缩放（可能由于此图形硬件不支持缩放功能并且软件缩放太慢），这时就只能通过名字选取对象。如果只是因为显示器太拥挤，那么用命令将对象名切换为可用或不可用就非常有用。

如果按照元素的含义给选项集合中的各元素命名，则采用键盘输入可允许我们使用通配符或禁忌字符，而做出多种选择。通过名字选择最适合于经验丰富、经常进行输入的用户，而不适合偶尔进行输入的用户。

如果要从键盘键入名字，则每次击键之后立即显示一条有用的反馈信息，将选项集合中与目前键入的字符序列相匹配的名字列表（或者部分列表，如果全部列表太长的话）显示出来。如果用户已经回想起前几个字符，这样显示出来名字列表有助于用户记得刚才是怎样拼写名字的。一旦键入一个惟一匹配的名字，就会自动在列表中高亮度显示正确的名字，或者自动完成名字尚未键入的部分。这种技术称为**自动完成**。新用户有时难以适应该技术，因而需要慎重使

用。另一种用于输入名字的策略是拼写校正（有时称Do What I Mean 或 DWIM），如果键入的名字与系统中已知的名字不匹配，其他与键入名相近的名字会作为可选名字呈现给用户。判断相似性可能与查找单字符错误一样简单，也可能有多个字符错误或者缺少字符。

用户可以不用击键而是借助语音识别器说出一个名字、缩略词或者代码。语音输入是从数据中区分命令的一种简单途径：通过语音输入命令，通过键盘或其他方式输入数据。在键盘环境中，语音输入消除了用特殊字符或其他特殊方式区分数据与命令的必要性。

2. 通过定位选择对象

8.2节引言部分提到的任何一种定位技术都可用于选择对象。对需点击的期望对象，先定位然后指示（一般通过按下按钮）。但是，如果该对象像第7章中的机器人那样具有多层结构又怎么办呢？如果光标停在机器人的手上方，就弄不清楚用户需点击手、胳膊还是整个机器人。可以使用Select_robot（选择机器人）和Select_arm（选择胳膊）之类的命令指明结构的层次。另一方面，如果用户工作所处的层次不经常改变，就可以用一条单独的命令，如Set_selection_level（设置选择层次），来修改结构的层次，这样工作速度更快。

如果系统设计者不知道结构的层次数目并且数目可能非常大（如绘图系统中，符号由图元和其他符号组成），这时就需要采用另外一种方法。至少需要两条用户命令：Up_hierarchy（向上一
层）和Down_hierarchy（向下一层）。用户选择某对象时，系统高亮度显示最低层的可见对象。如 306
果正是想要的对象，用户继续操作。如果不是，用户发出第一条命令：Up_hierarchy。高亮度显示的整个第一层对象包含已检查对象。如果这不是用户想要的，他就再一次向上遍历，图像的大部分仍是高亮显示。如果正处于最高一层，可以用Down_hierarchy命令沿相反方向向下选择对象。此外，Return_to_lowest_level（转到最低层）命令在深层次结构中非常有用。可以在另一个窗口的层次图中显示当前的选择位于哪一层。图8-4所示的状态图显示了层次选择的一种方法。另外，使用一条Move_up_hierarchy（移动到前一个层次）命令能够在到达节点后跳回到最初所选的叶节点。

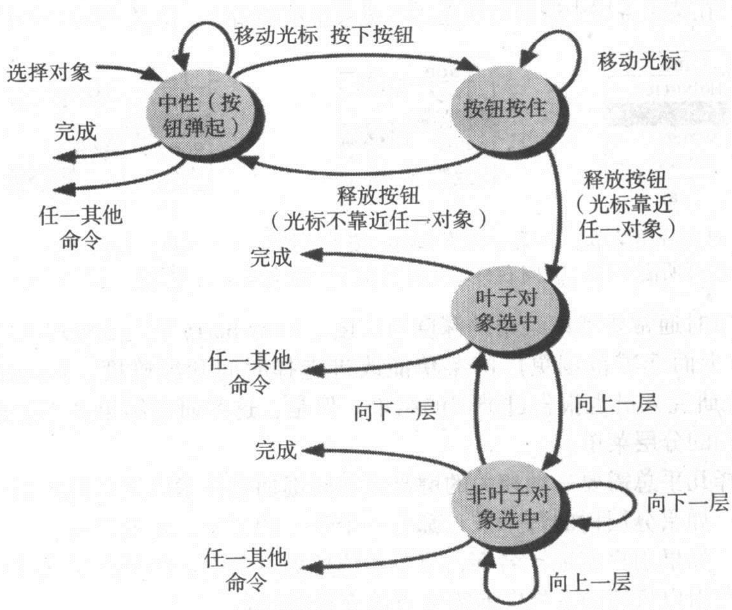


图8-4 用于具有任意层次对象-选择技术的状态图。Up和Down命令用于向上、向下层次移动，
“Leaf object selected”状态Down_hierarchy命令不可用。用户通过用光标定位对象、按下 307
按钮然后释放来选择对象

8.2.3 选择交互任务——相对固定大小的选项集合

菜单选择是一种从相对固定大小的选项集合中进行选择的最常用技术，这里讨论几个菜单设计中的关键因素。

1. 单层与多层设计

如果选项集合太大一次不能全部显示，就需要采用这种最基本的菜单设计方法。这样的菜单被细分成按逻辑结构组织的层次，或者将选项的线性序列分页显示或滚动显示。很多窗口管理器使用的滚动条允许以一种简洁的方式显示所有相关的滚动命令和分页命令，还可以提供一种快速定位滚动命令的面向键盘的方式。例如，可以用箭头键滚动窗口，也可将Shift键与箭头键组合使用，在可见窗口内部移动选项，如图8-5所示。

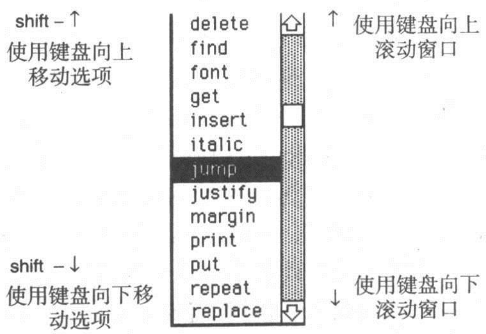


图8-5 滚动窗口内部的菜单。用户通过选择向上箭头键和向下箭头键或者通过拖动滚动条中的方块来控制菜单项的上下滚动

如果使用分层菜单，用户首先要从最高层的选项集合中选择，然后出现第二级选项集合，重复这一过程直到选择分层菜单树的叶节点（即选项集合本身的一个元素）。在进行分层对象选择时，需要提供导航机制，以便如果选择了不正确的子树，用户可以向上退回一个层次，同时需要提供视觉反馈信息以使用户能意识到所处的层次。

有多种方式显示菜单层次。随着菜单选项的一层层展开，由下一级菜单完全取代上一级菜单固然可以，但是不利于用户了解所处的菜单层次。图8-6描述的**级联式分层**菜单更具吸引力，每个菜单都必须充分显示，使用户可以看见完整的高亮度显示的选择路径，同时必须采用一些方法来说
308 明每个菜单项是一个叶节点还是下一级菜单的名字（图中的向右箭头充当这一角色）。另一种菜单分层方式是只显示在菜单各层中遍历时所选的各级菜单项的名字，以及当前层次中的所有选项。

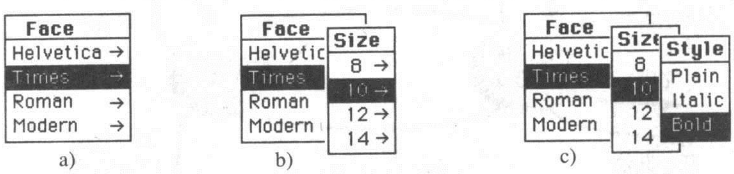


图8-6 一个弹出式分层菜单。a) 第一级菜单响应按下按钮操作显示出光标所处位置，可以上下移动光标选择所需字体；b) 向右移动光标弹出下一级菜单；c) 重复该过程弹出第三级菜单

设计分层菜单时通常要考虑菜单的深度与广度。Snowberry等人[SNOW83]通过实验发现，使用选择层次较少而选项范围更广的菜单能改进选择时间和准确度。Landauer和Nachbar [LAND85]及其他研究人员也报告过类似的结果。但是，这些研究结果并不能推广到缺乏自然性及可理解性结构的分层菜单。

分层菜单选择几乎总需要一种相关的键盘或功能键加速技术以提高熟练用户（即所谓的**高手**）的选择速度。如果分层树的每个节点都有一个惟一的名字，要做到这一点很容易，用户可以直接输入名字。如果用户忘记了名字，菜单系统还提供备份。如果在层次的每一级内部保持名字惟一性，熟练用户只须键入指向所需节点的完整路径名。

2. 菜单放置

显示在显示器屏幕上的菜单可以是静态的并且长久可见的，也可以按要求动态出现（如浮动菜单、隐含式菜单、弹出式菜单、下拉式菜单和拉出式菜单）。

弹出式菜单是在做出相应选择的时候出现在屏幕上的，它要么是响应一个明确的用户操作（典型地按下鼠标或输入板定标器的按钮），要么是由于下一个对话步骤要求进行菜单选择而自动出现。这类菜单一般显示在光标位置，这个位置通常是用户的视觉注意的中心，可以保持视觉连续性。弹出式菜单一个引人注意的特点是认为最近选择过的菜单项比其他项更有可能被再次选到，因此首先高亮度显示最近从选项集合中选择过的菜单项，将光标定位在该菜单项上。

弹出式菜单及其他隐含式菜单节省了宝贵的屏幕空间——用户界面设计者最宝贵的资源之一。第2章讨论的快速RasterOp指令促进了这些菜单的使用。 309

与弹出式菜单不同，下拉式菜单和拉出式菜单被定位在沿屏幕边缘排列的菜单条中。Apple Macintosh、Microsoft Windows、OPENLOOK和Motif都使用下拉式菜单。图8-7所示的Macintosh菜单还利用了快捷键和上下文敏感性。

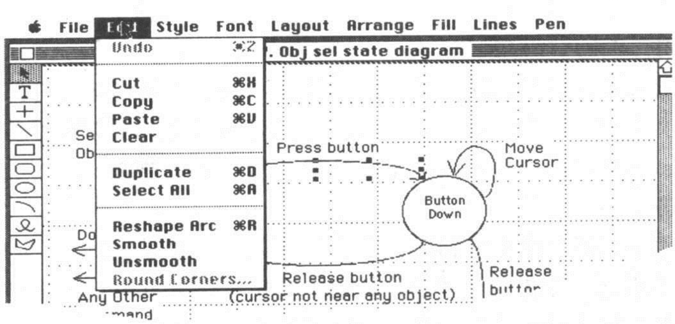


图8-7　一个Macintosh下拉式菜单。最后一个菜单项是灰色而不是黑色，这表示该项当前不可用（当前被选择的对象是一条圆弧，没有圆角）。Undo命令也被置为灰色，这是由于原先执行过的命令不可能被撤销。菜单中的缩略词是供熟练用户使用的快捷键。（Claris公司版权所有，1988，保留所有权利。）

3. 当前选择

如果一个系统有这么一个概念，即选项集合的"当前所选元素"，则菜单选择时允许该元素被高亮度显示。有些情况下，由系统提供最初的默认设置并且一直使用到用户修改了默认值。可以用多种方式显示当前所选元素，以汽车收音机上的调节按钮为模式的**单选按钮**（radio-button）交互技术即是其中一种（图8-8）。另外，一些弹出式菜单假定用户更有可能重选上次选过的菜单项，因此高亮度显示最近选择过的选项并且将光标停在该项上。

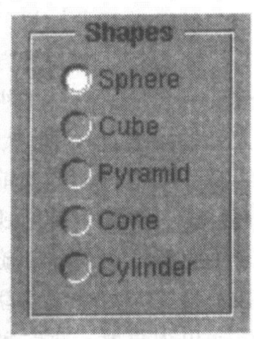


图8-8　用于从相互排斥的选项集合中进行选择的单选按钮技术。（NeXT公司授权，©1989 NeXT, Inc.。）

4. 菜单项的大小和形状

定位的精确性和速度受每个菜单项的大小的影响，Fitts法则指出，菜单项越大，选择速度就越快[FITT54；CARD83]，但另一方面，小的菜单项占用的空间更少并且允许在固定大小的区域中显示更多的菜单项，不过，在选择过程中产生的错误更多。因此，在使用小菜单项以节省屏幕空间方面与使用较大菜单项以减少选择时间和错误率方面存在着矛盾。

5. 模式识别

在涉及模式识别的选择技术中，用户使用连续定位设备（如输入板或鼠标）做连续运动。模式识别装置自动将运动序列与一系列已定义的模式相比较，每个模式对应于选项集合中的一个元素。 310
表示删除、大写、移动等校对用的标记特别适用于这种方法[WOLF87]。

最近，在字符识别算法上的进步已经导致了基于笔的操作系统和便携式膝上型计算机，如苹果公司的Newton，各种模式通过输入板输入，并被识别和解释成命令、数字和字母。

6. 功能键

可以将选项集合中的元素与功能键联系起来（如使用常规键盘上的单键作为功能键输入），但不幸的是没有那么多的键可用！各个键可用于分层选择模式中，还可以使用和弦修改它们代表的含义。比如说与功能键一起按下键盘上的Shift键和Ctrl键。不过，要学习用于一些命令的外部键组合（如Shift+Ctrl+L）并不容易。一般它们留做想提高效率的常规用户练习。在键盘上放一张练习用的模板，提示用户这些难记的组合键可以加快学习过程。也可以定义一些代表某种含义的组合方式来减少学习时间，例如，Macintosh上的Microsoft Word使用“Shift+ >”增大点的大小，用相应的“Shift+ <”减小点的大小；“Shift+I”将普通正体字设成斜体，或将斜体字设成非斜体，而“Shift+U”用类似方式处理带下划线的正文。

8.2.4 文本交互任务

文本串输入任务限定在应用系统中输入的字符串不赋予任何特殊意义，因此，输入一条命令的名字不是一项文本输入任务，相反，输入图形的说明以及向文字处理器中输入文本都是文本输入任务。显然，最常用的文本输入技术就是使用QWERTY键盘。

8.2.5 定量交互任务

定量交互任务用于在某个最小值和最大值之间指定一个数值，典型的交互技术有键入数值、用刻度来设置值、使用上下变化的计数器选择值。与定位任务类似，定量交互任务可以是语言的，也可以是空间的。如果是语言的，则用户知道输入的具体值；如果是空间的，则用户用某一增量来增加或减少一个值，从而得到的可能是最终想要值的近似。前一种情况下，交互技术显然必须包括所选值的数值反馈信息（获取反馈的一种方式是给用户以输入实际值）；后一种情况下，对数值的设置有一个大概印象更加重要，这一般用空间定向反馈技术来实现，如显示一个刻度盘或者标尺，其上显示当前（也可能是以前）的值。

输入数值的一种方式是使用电位器。决定使用旋转式还是直线式电位器需要考虑的因素是，改变一个值所产生的视觉反馈信息是旋转式的（如一个转动的时钟臂）还是直线式的（如一支上升的温度计）。虽然旋转式电位器有指针指示，而且旋转式电位器更容易调节，但一个或一
311 组滑动式电位器的当前位置比旋转式的更容易掌握。另一方面，旋转式电位器很容易调节。同时采用直线式电位器和旋转式电位器有助于用户将数值含义与每个设备联系起来，始终采用一致的方向也很重要：顺时针方向运动与向上运动通常应增加值的大小。

使用连续刻度操纵，用户可将光标定位在所示标尺的当前值指示器上，按下选择按钮，沿着标尺将指示器拖到期望数值上，然后释放选择按钮。通常使用指针指示标尺上所选的值，还可以提供数值回应。图8-9显示了这样几种刻度盘以及相关的反馈信息。

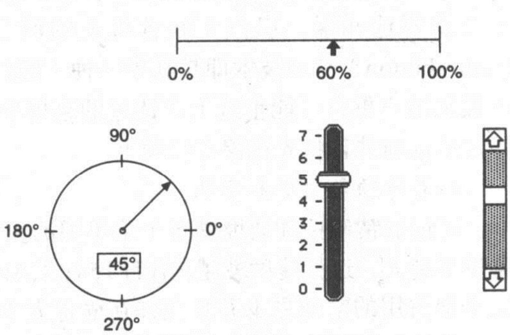


图8-9 用户通过拖动控制指针输入数值的几种刻度盘。通过指针和数字显示的两种方式来提供反馈信息。（垂直滑动尺由苹果计算机公司版权所有。）

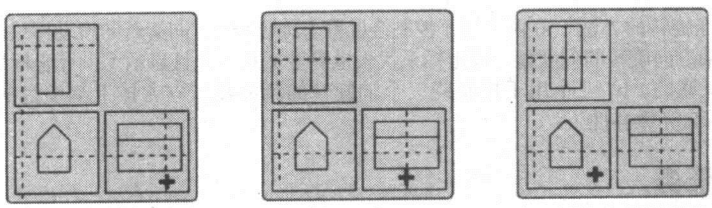
8.2.6 三维交互任务

前述用于二维应用系统的四种交互任务中有两种在三维应用中更加复杂：定位任务和选择任

务。本节的第一部分介绍了定位和选择技术（它们密切相关）。在这一节，我们还引入另外一种三维交互任务：旋转任务（用于在三维空间中给对象定向）。复杂的一个主要原因是难以理解光标或对象相对于其他显示对象的三维深度关系，这一点与二维交互完全相反。在二维交互过程中用户很容易理解光标是在某对象的上方、旁边还是正指向对象。复杂的一个次要原因是因为通常所用的交互设备（如鼠标和输入板）都只是二维设备，需要用某种方法将这些二维设备的运动情况映射到三维空间中。

与左、右眼视图相对应的立体眼镜有助于理解一般的深度关系，但是作为一种精确定位的方法其精确性很有限。有关将立体眼镜用于人眼的方法在第12章以及[HODG85]中有所讨论，
其他表现深度关系的方法在第12~14章讨论。 312

图8-10表现了一种三维定位的常用方法。在鼠标控制下，二维光标在三个视图之间自由移动，用户可以用按下按钮、拖动、释放按钮的操作序列，选择任一条三维光标的虚线并拖动该虚线。如果按下按钮事件紧挨着两条光标虚线的交点，则两条光标虚线都被选中，并随着鼠标一起移动。虽然这种方法要求用户必须同时在一种或两种维数的空间中工作，看起来有一定的限制，但是，将三维操纵任务分解成更简单的低维数任务有时候很有好处。使用多个视图有助于选择和定位：相互重叠的对象在一张视图中难以区分，但在另一张视图中可能不会相互重叠。



当二维光标位于三维光标虚线之上时按下按钮　　拖动三维光标，所有视图被相应更新　　释放按钮，二维光标不再控制三维光标

图8-10　对同一场景（一所房屋）使用三个视图的三维定位技术。二维光标（+）用于选择一条三维光标的虚线

在定位与选择方面，三维旋转需要考虑的事项包括理解深度关系、将二维交互设备映射到三维以及保证刺激–响应兼容性（S-R兼容性）⊖。一种容易实现的三维旋转技术提供滑动标尺或刻度表来控制围绕三个轴的旋转。S-R兼容性要求正常情况下三个轴应当位于屏幕坐标系中：向右是x方向，向上是y方向，向屏幕外（或屏幕内）是z方向[BRIT78]。当然，旋转的中心要么必须以单独步骤明确定义，要么必须隐含定义（典型的是以屏幕坐标原点、对象原点或对象中心为旋转中心）。如图8-11a所示，围绕屏幕的x轴和y轴旋转非常简单。随着滑动块的移动，与滑动块相关联的(x, y, z)坐标系被旋转以显示旋转的效果。通过添加一个用于z轴旋转的刻度盘可以很容易地将二轴旋转方法扩展到三个轴旋转（刻度盘比滑动尺更适用于刺激–响应兼容性）。图8-11b所示的立方体表面刻度盘的排列方式可以提供更强的刺激–响应兼容性，该立方
体清楚地指出每个刻度盘控制的轴。也可以用三维跟踪球代替这些刻度盘。 313

经常有必要将几项三维交互任务结合起来，这样，旋转运动需要一项被旋转对象的选择任务、一项旋转中心的定位任务和一项实际旋转的定向任务。指定一个三维视图可以被看成一项组合的定位（眼睛位于何处）、定向（眼睛如何定向）以及缩放（视图的范围，或有多少投影面

⊖ 人的因素原理说明系统对用户动作的响应必须在同一方向，并且响应的幅度应该与动作成正比。

被映射到视口）任务。我们可以将前面讨论过的一些技术结合起来创建这样一项任务，也可以通过设计一种鸟瞰（fly-around）功能来创建，观察者可以利用这种功能驾驶一架想像中的飞机围绕一个三维世界飞行。一般用螺距、卷轴、偏航角来控制，外加速度用来加速减速。利用鸟瞰的概念，用户需要一个总的视图（如二维平面视图）来指示想像中飞机的地面位置和航向。

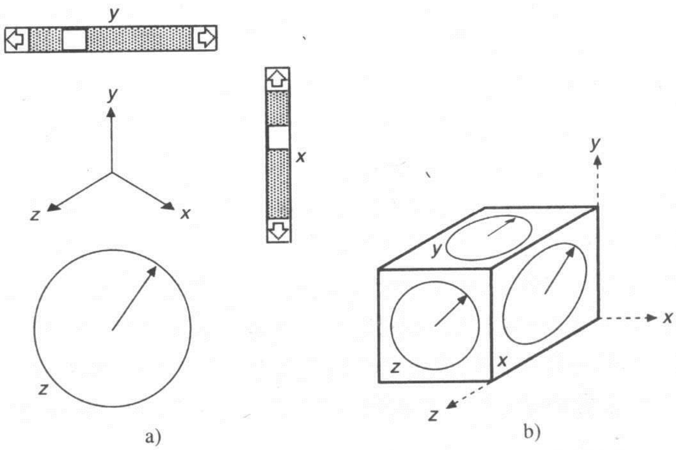


图8-11 两种三维旋转的方法。a) 两个用于控制绕屏幕上x轴和y轴方向旋转的滑动标尺与一个用于控制围绕 z 轴方向旋转的刻度盘，该坐标系代表世界坐标系，并且显示了世界坐标系如何与屏幕坐标系相关联的；b) 三个用于控制绕三个轴旋转的刻度盘，立方体上刻度盘的位置提供很强的刺激–响应兼容性

8.3 复合交互任务

复合交互任务(CIT)是基于上一节所介绍的基本交互任务而建立的，事实上，一个复合交互任务是若干个集成到一个单元的基本交互任务。复合交互任务主要有三种形式：(1) 对话框，用于指定多个信息单元；(2) 构造，用于创建需要两个或更多位置的对象；(3) 操纵，用于对已
314 经存在的几何对象进行重定形。

8.3.1 对话框

我们经常需要从一个选择集合中选取多个元素。例如，文本属性（如斜体、黑体、下划线、空心和全部大写）都不是互斥的，用户可能会同时选择其中的两个或更多个。另外，相关的属性可能组成几个集合，如字体与字型。在单选时有用的一些菜单技术不能满足多选的需要。例如，当做了一次选择之后，下拉式菜单和弹出式菜单通常都会消失，为了做第二次选择，你必须要再次激活它们。

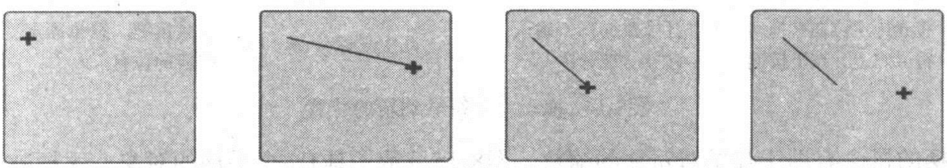
这个问题可以通过对话框来解决。对话框是一种菜单，它将一直保持可见，直到用户显式地关闭它。另外，对话框允许从多个选择集合中进行选择，并且它包含了一些区域，在这些区域中，用户可以输入文本或数值。在对话框中所做的选择可以立即被改正。当所有的信息都输入对话框之后，用户一般通过一个命令显式地关闭它。在对话框中指定的属性或其他值也可以立即被应用，让用户预览字体或线型等改变的效果。

8.3.2 构造技术

构造直线段的一种方法是让用户指定一个端点，然后指定另一个端点；当第二个端点被指定后，在两个端点之间画出直线段。然而，采用这种方法，用户很难在最终的直线段被画出之前尝试不同的线段位置，因为在指定第二个端点之前，直线段并没有实际被画出来。通过这种

交互方式，用户为了重新定位端点，每次都必须调用一个命令。

一个较好的方法是第2章讨论过的**橡皮筋线**技术。当用户按下按键（通常是输入板指示笔的按键或鼠标的按键）时，光标（一般由一个连续定位设备控制）建立了直线段的起始端点。随着光标的移动，直线段的终点也跟着移动。当按键释放时，直线段的端点被确定。图8-12显示了一个采用“橡皮筋线”技术的绘图顺序。注意，只有当按键被按住的时候，“橡皮筋线”状态才是激活的。也只有在这个状态下，光标的移动才会改变当前的直线段。

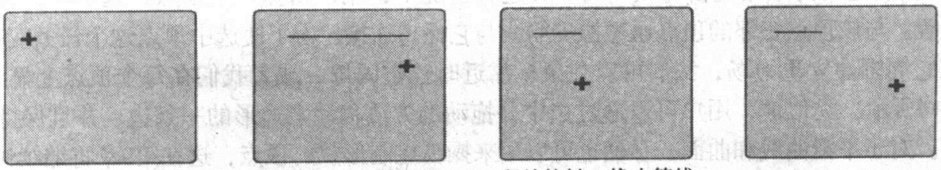


按下按钮，橡皮筋线从光标处开始　　从起始点到当前光标位置画一条直线段　　释放光标，橡皮筋线结束，直线段固定　　光标不再控制直线段

图8-12　橡皮筋线画线

从橡皮筋线绘图引申得到了一整套交互技术。**橡皮筋矩形**技术从一个按键动作开始，这个按键动作确定了矩形的一个角点，然后，它的对角点与光标动态地连接在一起，随着光标移动而移动，直到释放按键为止。这种技术的状态图与橡皮筋线技术的状态图之间的惟一区别是它们的动态反馈不同，一个是矩形，一个是直线段。**橡皮筋圆**技术创建一个圆，该圆的圆心由初始光标位置确定，该圆经过了当前光标位置，或者该圆在由对角确定的正方形的内部。如果外接矩形是正方形，得到的是圆。所有这些技术有共同的用户动作序列：按键、移动定位器并观察反馈、释放按键。

在以上任何一种技术中，不同的**约束**可以施加到当前的光标位置。例如，图8-13显示了使用与图8-12相同的光标位置产生的一列直线段，但是有水平约束。一条竖直线段，或者一条其他方向的直线段，也可以通过同样的方式绘出。全部由水平和竖直线组成的折线，就像印制电路板、VLSI芯片和一些城市地图中那样，很容易创建；直角可以由用户命令指定，也可以通过光标改变方向自动产生。这种思想可以推广到任意形状，如圆、椭圆或其他曲线。曲线从某个位置开始，然后通过光标的移动来控制曲线的哪些部分被显示。一般地，光标的位置被作为约束函数的输入，约束函数的输出被用来显示对象的适当部分。



按下按键，橡皮筋线从光标位置开始　　从起始点到x轴的新光标位置画线　　释放按键，橡皮筋线结束，直线段固定　　光标不再控制直线段

图8-13　用带水平约束的橡皮筋线绘图

8.3.3 动态操纵

仅仅创建直线段、矩形等图形是不够的，在许多场合，用户必须能够修改已经创建的几何实体。 315~316

如图8-14所示，在光标的控制下，拖动选定的符号从一个位置移动到另一个位置。通常按下按键动作开始拖动（在某些场合，按下按键也用来选定位于光标下将要被拖动的符号），然

后，一个释放按键的动作将符号固定在新的位置，随后的光标移动对符号没有影响。这个按下按键、拖动、释放的序列通常称为**点击-拖动**交互。

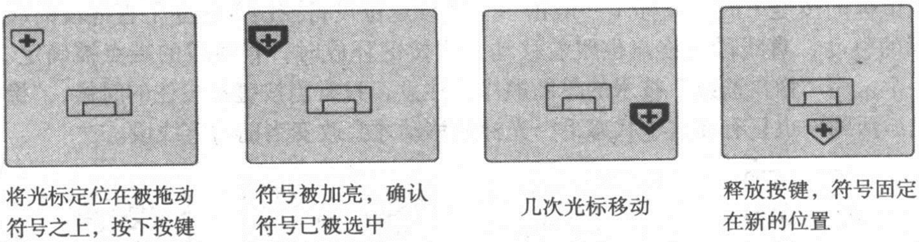


图8-14　拖动一个符号到新的位置

手柄的概念有助于提供对对象的缩放。图8-15显示了具有8个手柄的对象，手柄被显示成小的方块，它们分布于对象包围盒的角点和边界上。用户选择其中的一个手柄，然后通过拖动就可以缩放对象了。如果手柄位于一个角点上，那么对象的对角点的位置被锁定。如果手柄位于一条边界的中间，那么对象的对边的位置被锁定。

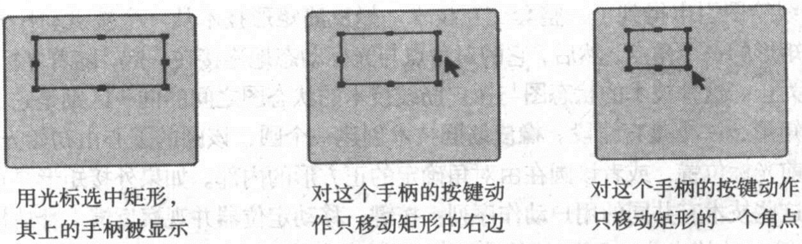


图8-15　用于改变对象形状的手柄

当将这种技术集成到一个完整的用户界面中时，只有对象被选中，其上的手柄才会被显示。同时，手柄也是惟一的表示对象被选中的视觉编码，因为其他视觉编码（如粗线段、虚线、亮度改变等）都有可能是图形对象本身的组成部分。

317

拖动、旋转和缩放影响整个图形对象，如果我们仅仅想改变单个的点，比如多边形的顶点，怎么办呢？可以给顶点命名，然后用户输入顶点的名称和它新的(x, y)坐标。但是，用于移动整个对象的指点-拖动的策略更富有吸引力。在这种情况下，用户指点一个顶点，选中它，拖动它到新的位置。与该顶点相邻的顶点通过橡皮筋线与它保持连接。为了使选中顶点这个任务变得容易，我们在它周围建立重力场，我们可以在鼠标靠近时让它闪烁，或者我们在每个顶点上显示手柄，如图8-16所示。类似地，用户可以通过选中并拖动的方法移动多边形的一条边，并且保持边的斜率不变。对于平滑曲线和曲面，手柄也可以用来操纵控制形状的顶点，这在第9章将继续讨论。

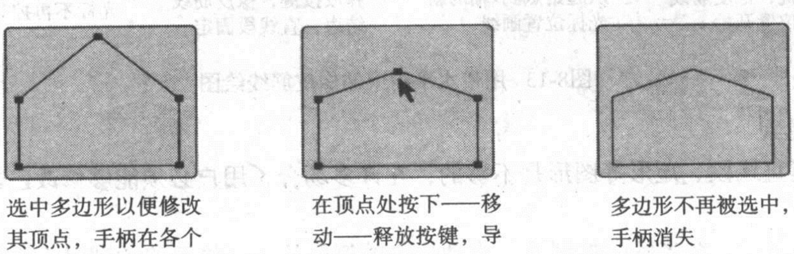


图8-16　手柄用于改变多边形的顶点

8.4 交互技术工具箱

用户-计算机界面的视感很大程度上由提供给它的交互技术集所决定。回想一下，交互技术实现了用户-计算机界面设计的硬件绑定部分。设计和实现一组好的交互技术很花费时间。作为交互技术的子程序库，交互技术工具箱是一种供应用程序开发人员使用的技术集合。这种方法有助于在应用程序中保证一致的视感，显然是一种有效的软件工程准则。

交互技术工具箱不仅可以被应用程序使用，而且也可以被窗口管理器使用，窗口管理器毕竟也是另一种客户端程序，在所有的应用程序中，使用同一个工具箱是一种重要而常用的方法， 318
它提供了与多种应用程序和窗口环境本身统一的视感。例如，用来选取窗口操作的菜单风格在应用程序中应保持一致。

工具箱可以在**窗口管理系统**[FOLE90]的顶层实现。在没有窗口系统的情况下，工具箱可以直接在图形子程序包的顶层实现；然而，由于工具箱的元素包括菜单、对话框、滚动条以及所有能在窗口中方便实现的类似的部件，所以通常使用窗口系统的底层。广泛使用的工具箱包括Macintosh工具箱[APPL85]、X Window System使用的OSF/Motif[OPEN89]和InterViews[LINT89]，以及实现OPEN LOOK的几个工具箱[SUN89]。彩图9所示为OSF/Motif界面，彩图10所示为OPEN LOOK界面。

小结

我们已经介绍了一些最重要的用户界面的概念：输入设备、交互技术和交互任务。用户界面技术和设计还有许多我们没有讨论的方面，其中有各种对话风格（如所见即所得（WYSIWYG）、命令语言和直接操纵）的优缺点以及影响用户界面的窗口管理器问题。[FOLE90]对这些主题有完整的介绍。

习题

8.1 检查一个你所熟悉的人机界面，列出用过的每一个交互任务，看看它们属于8.2节中4类基本交互任务的哪一类。如果一个交互任务不属于其中任何一类，试着将它进一步分解。

8.2 扩展图8-4的状态图，使之包含“返回到最低层”命令。该命令把选择层次返回到最低层，因此最先被选中的将再次被选中。

8.3 在具有查色表的彩色光栅显示器上实现一个菜单包，使得菜单以亮的但部分透明的颜色显示，而菜单下面用柔和的灰色显示。

8.4 实现本章讨论的任何一种三维交互技术。

8.5 画出弹出式分层菜单的状态图，画出面板式分层菜单的状态图。 319

第9章 曲线与曲面的表示

图9-1展示的经典茶壶或许是计算机图形学中最有名的图标。自从1975年Martin Newell [CROW87]用它构造模型以来，茶壶已经被大批研究者用做生成真实感曲面和纹理最新技术的一个演示结构样品。一个雅致的茶壶的建模，需要将它的形状指定为一组光滑的曲面元素（称为**双三次曲面片**）。许多计算机图形学的应用系统都要求生成光滑的曲线与曲面。许多现实中物体本身就是光滑的。对现实世界的模拟，正是计算机图形学所要研究的。计算机辅助设计（CAD）、高质量字符的字形、数据的绘图以及艺术家的素描等都包含大量的光滑曲线与曲面。在一段动画中，物体运动和视角变化的轨迹也几乎总是光滑的；类似地，物体亮度或颜色的变化通常也必须是光滑的（详见第12章和第11章）。 321

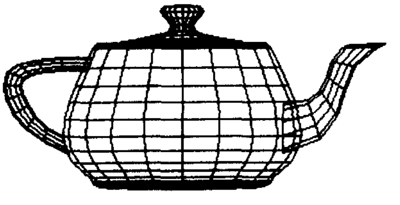


图9-1 著名的茶壶——一个由平滑曲面聚集而成的模型

在下面两种情况下，我们需要表示曲线与曲面：模拟已存在的物体（比如一辆汽车、一张脸、一座山等）和根据“草图”进行造型，对于后者，所构造的物体是原先不存在的。在第一种情况中很难找到一种对物体的数学描述。当然，我们可以把物体表示成无限多个坐标点的模型，但这对存储空间有限的计算机来说是不可能的。通常，我们只是通过一些平面、柱面或者其他在数学上很容易表示的形状来近似物体，并让我们的模型上的点尽可能地接近实际物体中相应的点。

在第二种情况中，用户在建模过程中直接构造出原先并不存在的物体，因此这个物体形状将与它在计算机中的表示完全吻合，因为表示只是物体形状的一种表现形式。构造物体时，用户可以交互地构造物体，用数学方法描述它，或者只给出一个大概描述而让某个专门程序去“填充”。在计算机辅助设计中，计算机设计出的抽象物体还要转化为现实的产品。

本章将对**曲面造型**的一般领域做一个总体的介绍。这一领域十分广泛，这里只对其中三种最常用的三维曲面的表示方法进行详述：多边形网格曲面、参数曲面和二次曲面。同时还讨论参数曲线，不仅因为其本身值得研究，而且因为参数曲面就是对参数曲线的一个简单推广。

第10章要介绍的**实体造型**是对体的表示，这个体被表面完全包围，比如一个立方体、一架飞机或一幢建筑物等。本章所讨论的曲面的表示方法可以用在实体造型中，去定义包围实体的每一个面。

多边形网格是由一系列彼此相连的多边形平面构成。像体可以被平面包围一样，打开的盒子、壁橱、建筑物外表等都可以用平面网格很容易并且很自然地表示出来。多边形网格也可以用来表示表面弯曲的物体，当然，这样的表示比起表示平坦表面的物体困难一些，并且只是近似表示，如图9-2所示。图9-3是用多给出了曲线形状的横截面和表示该形状的多边形网格。这种表示中的误差是显然的，但通过增加更多的多边形的边数，以得到更好的分段线性逼近，可以使误差任意小。但这样也同时增加了对存储空间和处理这种表示的算法执行所需的时间的要求。不仅如此，当图像被放大时，这些误差又会变得十分明显。

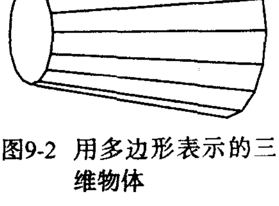


图9-2 用多边形表示的三维物体

参数多项式曲线使用以t为参数的三个多项式（分别对应x, y和z）来定义三维曲线上的点。选择多项式的系数，以便确定曲线的形状走向。虽然多项式的次数可以是任意的，但我们这里只讨论其中最常用的三次多项式（参数的最高次数为3）。用三次参数多项式表示的曲线通常称为**三次曲线**。

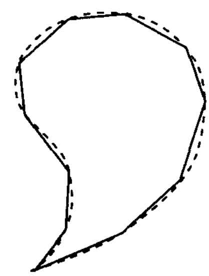


图9-3 弯曲物体（虚线）的横截面及其多边形表示（实线）

参数双变量多项式曲面片使用三个双变量多项式（分别对应x, y和z）来定义曲面上点的坐标。这样的曲面片的边界是参数多项式曲线。
322 在精度给定的条件下，用双变量多项式曲面片来表示弯曲表面比用多边形来表示所用的片数要少得多。但是前者所用的算法也要复杂得多。与曲线情形一样，多项式的次数可以任意，但我们这里只讨论最常用的三次多项式，即两个参数的最高次数为3。相应地，这样的曲面称为**双三次曲面**。

二次曲面是用二次方程$f(x,y,z)=0$隐式定义的，其中，f是关于x、y、z的二次多项式。对于我们熟知的球体、椭球体和柱体等，二次曲面是一种很方便的表示方法。

第10章介绍的实体造型将在系统中采用这些表示方法表示物体表面以及被表面包围的（实）体。本章所介绍的几种曲面的表示方法有时也结合起来使用，以围成一个三维实体。

9.1 多边形网格

多边形网格是由边和顶点构成的集合。这些多边形彼此相连，每一条边至少属于两个多边形。一条边连接两个顶点，一个闭合的边序列构成一个多边形。一条边可以同时属于两个相邻的多边形，一个顶点至少可以被两条边共享。多边形网格有几种不同的表示方法，它们各有利弊，应用程序的编写者必须从中选择最合适的表示方法。一个应用程序也可以在外部存储、内部结构以及与用户交互构造网格时采用几种不同的表示方法以适应不同需要。

在评价不同的表示方法时有两个基本的标准：时间与空间。对一个多边形网格的典型操作包括搜索一个顶点的所有邻边，搜索共用顶点或共用边的所有多边形，寻找一条边的两个顶点，寻找多边形的边，显示多边形网格，以及识别出表示中的错误（比如，缺少了一条边、一个顶点或一个多边形）。一般来说，多边形、顶点和边之间的关系越直接，操作的速度就越快，同时所要求的空间也就越大。Woo[WOO85]曾经分析了对一个多边形网格数据结构的九种基本存取操作和九种基本更新操作的时间复杂度。

在9.1.1节和9.1.2节将讨论与多边形网格相关的几个问题：多边形网格的表示，保证给定表示方法的正确性，以及多边形平面系数的计算。

9.1.1 多边形网格的表示

本节将讨论三种多边形网格表示法：直接表示，顶点表指针表示和边表指针表示。在**直接表示法**中，每一个多边形用一个顶点坐标序列来表示：

323
$$P=((x_1,y_1,z_1),(x_2,y_2,z_2),\ldots,(x_n,y_n,z_n))$$

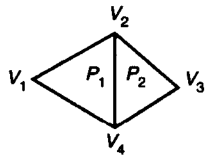
这些顶点按照沿多边形一周的顺序被存储，相邻顶点以及头尾顶点之间都包含一条边。对于单独一个多边形，这样表示的空间效率很高，但对于一个多边形网格来说，公共顶点坐标的重复浪费了大量空间，而且这样做也没有将公共顶点和公共边表示出来。比如，要拖动一个顶点和它的所有邻边，就必须先找到所有包含这个顶点的多边形，这需要将一个多边形的顶点坐标三元组与另外一个多边形的相比较。最有效的办法是将N个坐标的三元组排序，但这个过程的时间复杂度至少是$N\log_2 N$，甚至有时还会由于计算时舍入产生的误差，使在每个多边形中同一顶

点的坐标值不完全相等，这样就不可能得到正确的匹配。

在用这种表示法时，不管是用填充多边形还是用多边形边框来显示多边形网格，都必须先对每个顶点进行坐标变换并对每条边进行裁剪。在画边时，每条公共边会被画两次，这样的重画在笔式绘图仪、胶片记录器和向量显示器中就会产生问题。即使用光栅显示设备，当一条边从相反方向绘制两次时，也可能会出现肉眼可见的多余像素点。

SPHIGS中采用的是另一种表示法，用**顶点表指针**定义多边形，多边形网格中的每个顶点仅在顶点序列$V = ((x_1, y_1, z_1), \cdots, (x_n, y_n, z_n))$中被存储一次，用一个指向这个顶点序列的索引表（指针表）来定义多边形。若一个多边形由顶点3，5，7和10构成，则它被表示为$P = (3, 5, 7, 10)$。

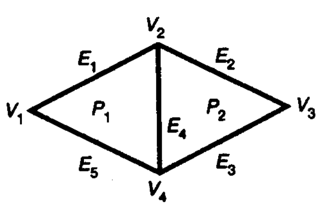
图9-4有一个顶点表指针表示法的例子，这种表示法与直接表示相比有几个优点。因为每个顶点只被存储一次，节省了大量空间，而且这些顶点的坐标可以很方便地进行修改。但是另一方面，这种方法还是没有解决搜索具有公共边的多边形和公共边被画两次这两个问题。下面介绍的方法通过边的直接表示，比较好地解决了这两个问题。



$$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \cdots, (x_4, y_4, z_4))$$
$$P_1 = (1, 2, 4)$$
$$P_2 = (4, 2, 3)$$

图9-4 用顶点的索引表定义的多边形网格

用**边表指针**定义一个多边形时，还是要有一个顶点序列V，但表示多边形的指针不是指向顶点序列，而是指向一个边序列。在这个边序列中，每条边只出现一次，边序列中的每条边都指向定义该边的顶点序列中的两个顶点，还指向一个或两个该边所属的多边形。这样，一个多 324
边形描述为$P = (E_1, \cdots, E_n)$，一条边为$E = (V_1, V_2, P_1, P_2)$。若一条边只属于一个多边形，则P_1或P_2为空值。图9-5给出了一个这种表示法的例子。



$$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \cdots, (x_4, y_4, z_4))$$
$$E_1 = (V_1, V_2, P_1, \lambda)$$
$$E_2 = (V_2, V_3, P_2, \lambda)$$
$$E_3 = (V_3, V_4, P_2, \lambda)$$
$$E_4 = (V_4, V_2, P_1, P_2)$$
$$E_5 = (V_4, V_1, P_1, \lambda)$$
$$P_1 = (E_1, E_4, E_5)$$
$$P_2 = (E_2, E_3, E_4)$$

图9-5 用每个多边形的边表定义的多边形网格（λ代表空值）

通过显示所有的边，而不需要显示每一个多边形，就能显示出整个多边形网格的框架，这就避免了冗余裁剪、变换、扫描转换等复杂操作。同样，也容易显示填充的多边形网格。对某些情形，比如三维的蜂窝状钣金结构的表示，一些边由三个多边形共享。这时，可以对边的描述信息进行扩展，使其包含任意多个多边形$E = (V_1, V_2, P_1, P_2, \cdots, P_n)$。

在这三种表示法（直接表示、顶点表指针、边表指针）中，要找到与一个顶点相邻的边，都比较困难：因为要检查所有的边。当然，可以直接添加一些信息来确定点边之间的关系。例如，Baumgart[BAUM75]使用了翼边表示，其中边的描述增加了指向每个多边形中两条邻边的

指针，同时顶点描述增加了指向与该顶点相接的（任意）一条邻边的指针，这样，边的描述就包括了更多的多边形与顶点的信息。

9.1.2 平面方程

在处理多边形或多边形网格时，我们经常需要知道多边形所在的平面的方程。当然，在某些情况下，用交互方法定义多边形时，平面方程是已知的，如果该方程未知，可以用三个顶点的坐标确定一个平面。平面方程为

325 $$Ax + By + Cz + D = 0 \tag{9-1}$$

系数A，B和C确定了平面的法向量$[A\ B\ C]$。若已知平面上的三个点P_1，P_2，P_3，就可以通过向量叉乘$P_1P_2 \times P_1P_3$（或者$P_2P_3 \times P_2P_1$，等等）计算出平面的法向量。如果这个叉乘为零，那么这三点共线，不能确定一个平面。可能的话，可以用另外一个顶点来代替。由这一非零叉积，将向量$[A\ B\ C]$和三点中的任意一个的坐标代入式(9-1)即可得到D。

如果有三个以上的顶点，由于数值误差或产生多边形的方法所产生的误差，这些顶点可能不在同一平面上。那么就需要另一种技术来确定方程系数A，B，C以使这个平面最接近于所有的顶点。可以证明，A，B和C分别与多边形在(y, z)，(z, x)和(x, y)平面上的投影面积成正比。比如，若多边形与平面(x, y)平行，那么$A = B = 0$，因为多边形在(y, z)和(z, x)平面上的投影面积分别为零。这一方法的优点是投影面积是所有顶点坐标的函数，从而与顶点的选取是否恰好共线或共面的情况关系不大。举例来说，如图9-6所示，多边形在(x, y)平面上投影的面积（从而也是系数）C等于梯形A_3的面积减去A_1和A_2的面积。一般地，

$$C = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i\oplus 1})(x_{i\oplus 1} - x_i) \tag{9-2}$$

其中操作符$\oplus$除$n\oplus 1 = 1$外是正常的加号。A和B的面积也可以用类似的公式求出，注意B的面积是负的（见例9.1）。

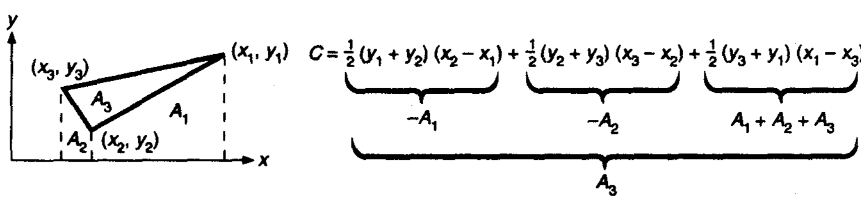


图9-6 利用式(9-2)计算三角形的面积C

由式(9-2)得到所有由多边形的相邻边构成的梯形面积的总和。若$x_{i\oplus 1}<x_i$，则这块面积为负数。总和的符号也是有用的：若顶点是按顺时针方向排列(投影到平面上时)，则符号为正，否则为负。

326 一旦用所有顶点坐标得到了平面方程，我们可以通过计算每个顶点到平面的垂直距离来估计出这个多边形不共面的程度。从顶点(x, y, z)到平面的距离d为：

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \tag{9-3}$$

这个距离可以是正数或者负数，取决于点在平面的哪一侧。若顶点落在平面上，则$d = 0$。当然，若只需判断点在平面的哪一侧，只要d的符号就够了，所以就没必要除上那个平方根。

平面方程并不是惟一的；方程两边可以都乘上一个常数k，表示的还是同一个平面。通常

把平面系数规范后再存储，可以将k设为

$$k=\frac{1}{\sqrt{A^2+B^2+C^2}} \tag{9-4}$$

即法向量长度的倒数。这样用式(9-3)计算距离就更容易了，因为分母为1。

例9.1

问题：给定一个多边形（近似地是平坦的）的n个顶点写一个计算平面方程系数的函数。假定从平面的正面观察该平面时，多边形的顶点是顺时针排列的。顶点及其数目是该函数的输入参数。

解答：使用式(9-2)以及改为A和B的类似算式，计算平面方程系数的函数的程序是简单的：

```
FindPlaneCoefficients(float x[], float y[], float z[], int num_verts,
                      float *a, float *b, float *c, float *d)
{
    float  A, B, C, D;
    int    i, j;

    A = B = C = 0.0;
    for (i = 0; i < num_verts; i++) {
        j = (i + 1) % num_verts;
        A += (z[i] + z[j]) * (y[j] − y[i]);
        B += − (x[i] + x[j]) * (z[j] − z[i]);
        C += (y[i] + y[j]) * (x[j] − x[i]);
    }
    A /= 2.0; B /= 2.0; C /= 2.0;
    D = −(A * x[0] + B * y[0] + C * z[0]);

    *a = A;
    *b = B;
    *c = C;
    *d = D;
}
```

327

9.2 三次参数曲线

曲线与曲面可以分别用折线和多边形进行一次线性分段逼近。除非被逼近的曲线与曲面也是分段线性的，否则为了达到一定的精度，就要生成并存储大量的顶点坐标。由于大量的点要精确给定，从而在做形状逼近时，使数据的交互控制变得繁琐。

本节要介绍一种结构更紧凑、更易于控制的分段光滑曲线的表示方法；在9.3节，这一方法还将推广到曲面情形。一般的方法就是用比线性函数更高次的函数来表示形状，这种函数仍然只能是近似表示，但比线性函数占用更少的存储空间，并提供更灵活的交互操纵。

这种较高次数的逼近基于以下三种方法。第一种方法是将y和z直接表示成x的显函数，即$y=f(x)$，$z=g(x)$。这种方法的困难在于：(1)由一个x值不能得到多个y值，所以像圆、椭圆这样的曲线必须用几段曲线表示；(2)这一定义不是旋转不变的（描述一个曲线旋转需要很大的工作量，而且可能需要将一段曲线分成若干段）；(3)描述具有与坐标轴垂直的切线的曲线是很困难的，因为这种方法不能表示无穷大的斜率。

第二种方法是用一个形如$f(x, y, z)=0$的隐式方程的解来表示曲线，但它也有其自身的缺点。首先，给定方程的解可能比我们想要的多。举例说，描述一个圆时，用方程$x^2+y^2=1$最好不过了，但一个半圆又怎么描述呢？为此，必须加上一些原隐式方程所不包含的限制条件，比

如$x \geq 0$。另外，用隐式方程定义的曲线段在做连接时，很难确定它们的切线方向在连接点上是否相等，而在很多应用中要求切向连续。

这两种数学形式的共同优点是很容易判断点是否在曲线上或在曲线的哪一侧，正如我们在第3章中所看到的那样。曲线的法线也不难计算。在9.4节中我们还会讨论这种隐式方法。

第三种方法是曲线的**参数表示**，即$x = x(t)$，$y = y(t)$，$z = z(t)$，它解决了函数表示和方程隐式表示引发的问题，同时还具有其他很多优点。参数曲线用参数切向量（不会等于无穷大）代替几何斜率（可能为无穷大）。这里，曲线用**分段多项式曲线**逼近，而不用上节所用的分段线性曲线段。曲线的每一曲线段Q由三个函数定义：x，y和z，其中x，y和z分别是关于参数t的三次多
328 项式。

三次多项式最常用，因为低于三次的多项式在控制曲线形状时不够灵活，而高于三次的多项式又会增加不必要的摆动和更多的计算量。低于三次的多项式不能表示一条通过（插值）两端点并指定其端点处切向的曲线段。给定有四个系数的三次多项式，可以用四个已知条件求得未知系数。这四个条件可以是两个端点以及端点处的导数。类似地，一次多项式（直线）的两个系数由两个端点确定。对于直线来说，端点处的导数由直线本身决定而不能随意变动。平方（二次）多项式有三个系数，只要两个端点再加上另外一个条件，比如斜率或者第三个点，就能确定下来。

另外，三次参数曲线是三维空间中次数最低的非平面曲线。事实上，只要三个点就能完全确定一个二次多项式的三个系数，而这三点也确定了多项式曲线所在的平面。

定义高次曲线需要更多的条件，这样在交互生成时会造成曲线的摆动而难以控制。虽然如此，在一些应用中还是需要使用高次曲线，比如，汽车和飞机的设计，通过对高阶导数的控制来设计出符合空气动力学的曲面。事实上，参数曲线和曲面的数学表达式通常可以是任意次数n，而在本章n为3。

9.2.1 基本特性

定义曲线段$Q(t) = [x(t) \ \ y(t) \ \ z(t)]^{\mathrm{T}}$的三次多项式形如

$$
\begin{aligned}
x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\
y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\
z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \qquad 0 \leqslant t \leqslant 1
\end{aligned}
\tag{9-5}
$$

处理有限曲线段时，不失一般性，可以将参数t限制在$[0,1]$区间上。

令$T = [t^3 \ \ t^2 \ \ t \ \ 1]^{\mathrm{T}}$，并把三个多项式的系数矩阵定义为

$$
C = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \tag{9-6}
$$

就可以把方程(9-5)写为

$$
Q(t) = [x(t) \ \ y(t) \ \ z(t)]^{\mathrm{T}} = C \cdot T \tag{9-7}
$$

329 这是对方程(9-5)的一个简洁表示。

图9-7给出了两条相接的三次参数曲线段及其多项式，图中还说明了虽然参数多项式本身是单值的，但对于一个x值，可以有多个y值与之对应。（曲线的这张图同本节中所有图一样给出的都是用$[x(t) \ \ y(t)]^{\mathrm{T}}$表示的二维曲线。）

1. 曲线段之间的连续性

$Q(t)$的导数表示曲线的**切向量**。由式(9-7)，得到

$$\frac{d}{dt}Q(t) = Q'(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t) \quad \frac{d}{dt}z(t)\right]^{\mathrm{T}} = \frac{d}{dt}C \cdot T = C \cdot [3t^2 \quad 2t \quad 1 \quad 0]^{\mathrm{T}}$$
$$= [3a_xt^2 + 2b_xt + c_x \quad 3a_yt^2 + 2b_yt + c_y \quad 3a_zt^2 + 2b_zt + c_z]^{\mathrm{T}} \tag{9-8}$$

如果两条曲线段拼接成一条曲线，就称这条曲线具有G^0**几何连续**。若两条曲线段在拼接点处的切向量方向相同(大小不一定等)，则称曲线具有G^1几何连续，计算机辅助设计常常要求曲线段之间具有G^1连续。G^1意味着曲线段在拼接点处的几何斜率相同。若两个切向量TV_1和TV_2方向相同，那么其中一个必定是另一个乘以一个倍数：$TV_1 = k \cdot TV_2$，其中k>0[BARS88]。

若两条曲线段在拼接点处的切向量相等(方向和大小都相同)，称曲线具有关于参数t的一阶连续，或者说**参数连续**，表示为C^1。若拼接点处的n阶导数$d^n/dt^n[Q(t)]$的方向和大小都相同，则称曲线是C^n**连续的**。图9-8显示了三种具有不同阶的连续性的曲线。注意参数曲线段本身是处处连续的，这里关心的连续性指的是拼接点处。

切向量$Q'(t)$表示了曲线上一个点相对于参数t的速度，而$Q(t)$的二次导数则表示了加速度。假设一台照相机沿着三次参数曲线每隔相同时间间隔记录一张图片，则切向量给出了照相机沿曲线的速度。为了避免在最后的动画序列中出现突然的跳跃，照相机在拼接点前后的速度必须连续。在图9-8中，正是由于通过拼接点时的加速度的连续性，使曲线C^2与C^1在绕到终点前伸得更远。

一般来说，C^1连续蕴含了G^1连续，但逆命题一般不成立。也就是说，C^1连续性比G^1更强，所以具有G^1的曲线可以不具有C^1。然而，具有G^1连续的拼接点与具有C^1的拼接点看起来光滑度也差不多，如图9-9所示。

参数曲线的坐标系中的表示与一般函数很不一样，后者的自变量标在x轴上，因变量标在y轴上，而在参数曲线的坐标系表示中，自变量t根本就不标出来，这意味着不能仅从参数曲线坐标系中确定曲线的切向量。可以确定切线的方向，但不能确定其大小。若$\gamma(t)$，

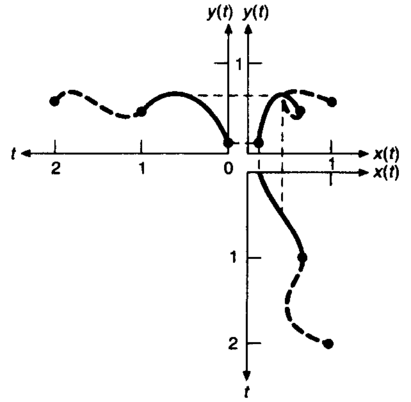


图9-7 两条相接的平面参数曲线及其多项式。(x,y)象限和$x(t)$以及$y(t)$象限之间的虚线表示曲线上的点(x, y)与相应的三次多项式之间的对应关系。第二段曲线的$x(t)$和$y(t)$经过参数变换后不从$t = 0$，而从$t = 1$开始，这样可以显示出连接点处的曲线连续性 330

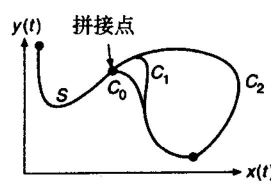


图9-8 曲线段S与曲线段C_0、C_1和C_2在拼接点处分别是0阶、1阶和2阶参数连续。C_1和C_2在拼接点附近差别很小，而远离拼接点时就有明显差异

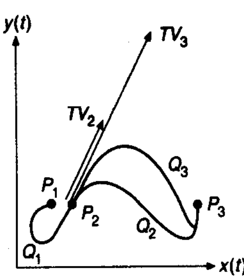


图9-9 曲线段Q_1、Q_2、Q_3在点P_2相接，除了P_2附近的切向量外其余条件都相同。Q_1和Q_2的切向量相等，因而它们在P_2处具有G^1和C^1连续。Q_3和Q_1切向量方向相同，但大小为Q_1的两倍，所以它们在P_2只有G^1连续。切向量越大，曲线Q_3在到达P_3之前沿切向量方向拉伸得越远。向量TV_2是Q_2的切向量，TV_3是Q_3的切向量

$0 \leq t \leq 1$是一条参数曲线，它在时刻0的切向量是$\gamma'(0)$。如果令$\eta(t)=\gamma(2t)$、$0 \leq t \leq 1/2$，则γ和η在参数坐标系中相同。另一方面，$\eta'(0)=2\gamma'(0)$，因此，坐标系相同的两曲线可以有不同的切向量。这就是为什么要定义**几何连续性**的原因：为了两条曲线的光滑拼接，只要求它们的切向量方向相同，而不要求大小相等。

2. 与约束的关系

一个曲线段$Q(t)$可以用端点、切向量和曲线段之间连续性等约束条件来定义。由式(9-5)表示的三次多项式有四个系数，所以需要四个约束条件来列出四个方程，然后求解得到。本节主要讨论的曲线有：用两个端点以及两个端点上的切向量定义的Hermite曲线；用两个端点和另外两个控制端点切向量的点定义的Bézier曲线；以及几种由四个控制顶点定义的**样条**曲线。样条曲线在拼接点处具有C^1和C^2连续性，并靠近它们的控制顶点，但一般不插值这些点。样条曲线的类型有均匀B样条和非均匀B样条等。

考察怎样用四个约束条件确定式(9-5)中的系数，前面将三次参数曲线定义为$Q(t)=C \cdot T$。把系数矩阵改写成$C=G \cdot M$，其中M是4×4的**基矩阵**，G是几何约束的四个元素的矩阵，称之
331 为**几何矩阵**。几何约束就是指限定曲线的一些条件，比如端点或切向量等。用G_x表示由几何矩阵中的x分量组成的列向量，G_y和G_z有类似的定义。对于不同类型的曲线，可能M不同，也可能G不同，或者两者都不同。

M与G中的元素都是常数，所以乘积$G \cdot M \cdot T$是三个关于t的三次多项式。将积$Q(t)=G \cdot M \cdot T$展开得到

$$Q(t)=\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}=[G_1 \quad G_2 \quad G_3 \quad G_4]\begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix}\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{9-9}$$

我们可以用第二种方式来读这个方程：点$Q(t)$是几何矩阵G的列的加权和，每个列代表三维空间中的一个点或一个向量。

仅仅将$x(t)=G_x \cdot M \cdot T$展开，得到

$$\begin{aligned} x(t)=&(t^3m_{11}+t^2m_{21}+tm_{31}+m_{41})g_{1x}+(t^3m_{12}+t^2m_{22}+tm_{32}+m_{42})g_{2x} \\ &+(t^3m_{13}+t^2m_{23}+tm_{33}+m_{43})g_{3x}+(t^3m_{14}+t^2m_{24}+tm_{34}+m_{44})g_{4x} \end{aligned} \tag{9-10}$$

式(9-10)强调曲线是几何矩阵中各元素的加权和。每一个权是关于t的三次多项式，称为**调配函数**。调配函数B由$B=M \cdot T$得到。注意到与分段线性逼近的相似性，分段线性逼近只要两个几何约束（线段的两个端点），每个曲线段是由端点G_1和G_2定义的直线段：

$$\begin{aligned} x(t)&=g_{1x}(1-t)+g_{2x}(t) \\ y(t)&=g_{1y}(1-t)+g_{2y}(t) \\ z(t)&=g_{1z}(1-t)+g_{2z}(t) \end{aligned} \tag{9-11}$$

三次参数曲线其实就是直线段逼近的推广。三次曲线$Q(t)$是几何矩阵的4个列的组合，正像直线段是两个列向量的组合一样。

为了计算基矩阵M，现在介绍几种特殊类型的三次参数曲线。

9.2.2 Hermite曲线

Hermite曲线（以数学家Hermite的名字命名）由端点P_1、P_4以及端点处的切向量R_1、R_4上

的约束确定。(不用下标1、2而用1、4是为了与后面几节的表示一致，后面几节在定义曲线时将用中间点P_2和P_3代替切向量。)

Hermite基矩阵M_H建立了**Hermite几何向量**G_H和多项式系数之间的联系，为计算Hermite基矩阵M_H，由四个约束条件建立四个关于未知多项式系数的方程，然后求解未知数。 332

将Hermite几何矩阵的x分量G_{H_x}定义为

$$G_{H_x} = [P_{1_x} \quad P_{4_x} \quad R_{1_x} \quad R_{4_x}] \tag{9-12}$$

根据式(9-5)和式(9-9)，把$x(t)$改写为

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = C_x \cdot T = G_{H_x} \cdot M_H \cdot T = G_{H_x} \cdot M_H \ [t^3 \ t^2 \ t \ 1]^T \tag{9-13}$$

直接代入式(9-13)得$x(0)$和$x(1)$上的约束为

$$x(0) = P_{1_x} = G_{H_x} \cdot M_H \ [0 \ 0 \ 0 \ 1]^T \tag{9-14}$$

$$x(1) = P_{4_x} = G_{H_x} \cdot M_H \ [1 \ 1 \ 1 \ 1]^T \tag{9-15}$$

就像一般情况下对式(9-7)求导得到式(9-8)一样，现在对式(9-13)求导得到$x'(t) = G_{H_x} \cdot M_H[3t^2 \ 2t \ 1 \ 0]^T$。因而，切向量约束的方程可写为

$$x'(0) = R_{1_x} = G_{H_x} \cdot M_H \ [0 \ 0 \ 1 \ 0]^T \tag{9-16}$$

$$x'(1) = R_{4_x} = G_{H_x} \cdot M_H \ [3 \ 2 \ 1 \ 0]^T \tag{9-17}$$

四个约束方程(9-14)、式(9-15)、式(9-16)和式(9-17)可以写成矩阵形式

$$[P_{1_x} \ P_{4_x} \ R_{1_x} \ R_{4_x}] = G_{H_x} = G_{H_x} \cdot M_H \cdot \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{9-18}$$

要满足这个方程（以及相应的y和z的方程），M_H必须是式(9-18)中4×4矩阵的逆矩阵

$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \tag{9-19}$$

M_H是惟一确定的，将它代入$x(t) = G_{H_x} \cdot M_H \cdot T$，得到基于几何向量$G_{H_x}$的$x(t)$。类似地，$y(t) = G_{H_y} \cdot M_H \cdot T$，$z(t) = G_{H_z} \cdot M_H \cdot T$，从而有

$$Q(t) = [x(t) \ \ y(t) \ \ z(t)]^T = G_H \cdot M_H \cdot T \tag{9-20}$$

其中G_H是列向量

$$[P_1 \ \ P_4 \ \ R_1 \ \ R_4]$$

将$Q(t) = G_H \cdot M_H \cdot T$中的$M_H \cdot T$展开，就得到了**Hermite 调配函数**$B_H$，用多项式作为权因子对几何矩阵中各元素做加权和得到：

$$\begin{aligned} Q(t) &= G_H \cdot M_H \cdot T = G_H \cdot B_H \\ &= (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4 \end{aligned} \tag{9-21}$$

图9-10给出了四个调配函数。注意，当$t = 0$时，只有标有P_1的函数不为零：只有P_1能影响曲线在$t = 0$时的值，当t大于零，R_1，P_4和R_4对曲线形状有影响。图9-11给出了四个由几何向量

中y分量加权的调配函数、这些调配函数的加权和$y(t)$以及曲线$Q(t)$。

图9-12是一组Hermite曲线。它们之间的惟一区别是切向量R_1的长度，而切向量的方向是一定的。切向量越长，对曲线的影响也就越大。图9-13是另一组Hermite曲线，其中的切向量长度一定，但方向不同。在交互式的图形系统中，用户通过控制端点和切向量来得到所需要的形状。图9-14是实现这类交互的一种方式。

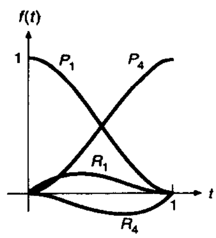


图9-10 Hermite 调配函数，标号表示此调配函数加权的几何向量中相应的分量

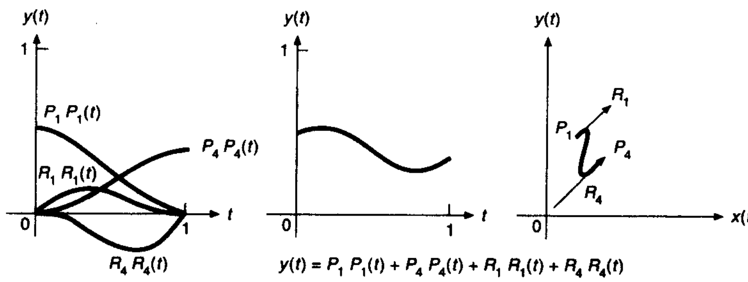


图9-11 Hermite曲线表示几何向量的四个分量与相应的调配函数的乘积(最左边的四条曲线)、加权和$y(t)$以及二维曲线本身(最右边)。$x(t)$也可由类似的加权和定义

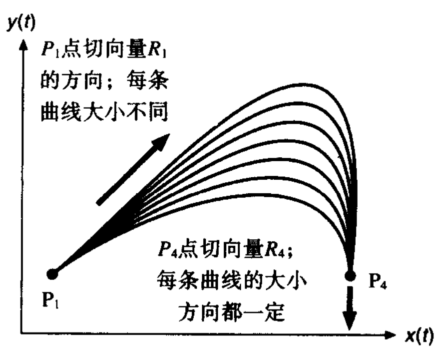


图9-12 一组Hermite三次参数曲线，其中只有在P_1处曲线的切向量R_1大小不同，曲线R_1的大小越大，曲线越高

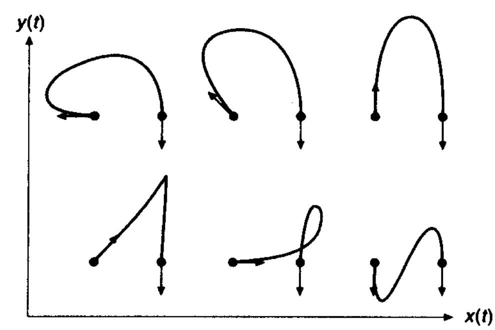


图9-13 一组Hermite三次参数曲线。只有左端点的切向量方向不同；所有切向量大小都相同。减小切向量的大小可以清除曲线的圈

画参数曲线

Hermite曲线和其他类似的三次参数曲线的显示较为简单：给定步长δ，取n个连续的t值，计算式(9-5)。程序9-1是显示曲线的源代码。在{ }之间的计算对每个三维点坐标有12步乘法运算和10步加法运算。利用Horner法则分解多项式化，

$$f(t) = at^3 + bt^2 + ct + d = ((at + b)t + c)t + d \tag{9-22}$$

将每一三维坐标点的计算稍稍简化为10步乘法和10步加法。

显式这些曲线的更有效的方法涉及前向差分技术，在[FOLE90]中有相应的讨论。

从式(9-10)可以看出，三次曲线是几何向量中四个分量的线性组合（加权和），因此可以通过变换几何向量来变换曲线，也就是说曲线在旋转、缩放和移动下不变。与把曲线变成一系列短直线段然后变换每个直线段以生成变换曲线相比，这种策略更为有效。但曲线在透视投影下

不具有不变性，这在9.2.6节还会讨论。

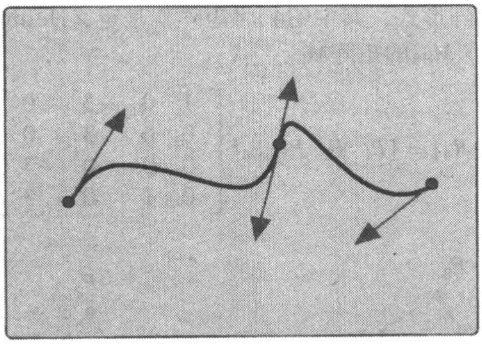


图9-14 两条Hermite三次曲线段，交互控制较为方便。端点可以通过拖动点来重新定位，拖动箭头可以改变切向量。拼接点处的切向量要求共线（以保证C^1连续性）：通常可以由用户给出命令指定C^0，C^1，G^1或不连续。为了显示清晰和方便用户交互，每条曲线的$t = 1$端点的切向量为相反方向画出

程序9-1 显示三次参数曲线的程序

```
typedef float CoefficientArray[4];
void DrawCurve(CoefficientArray cx, CoefficientArray cy,
               CoefficientArray cz, int n)
    /* cx, cy和cz是x(t)，y(t)和z(t)的系数 */
    /* 例如，Cx = GxM等 */
    /* n是步数 */
{
    float x, y, z, delta, t, t2, t3;
    int i;

    MoveAbs3( cx[3], cy[3], cz[3] );
    delta = 1.0 / n;
    for (i = 1; i <= n; i++) {
        t = i * delta;
        t2 = t * t;
        t3 = t2 * t;
        x = cx[0] * t3 + cx[1] * t2 + cx[2] * t + cx[3];
        y = cy[0] * t3 + cy[1] * t2 + cy[2] * t + cy[3];
        z = cz[0] * t3 + cz[1] * t2 + cz[2] * t + cz[3];
        DrawAbs3( x, y, z );
    }
}
```

9.2.3 Bézier曲线

三次多项式参数曲线段的Bézier形式[BEZI70；BEZI74]，以Pierre Bézier的姓氏命名，他开发了三次多项式参数曲线段用于在Rénault的汽车设计。它通过给定两个不在曲线上的中间点来间接地确定端点切向量，如图9-15所示。向量P_1P_2和P_3P_4决定了始点和终点处的切向量R_1和R_4，有关系式

$$R_1 = Q'(0) = 3(P_2 - P_1),\ R_4 = Q'(1) = 3(P_4 - P_3) \tag{9-23}$$

Bézier曲线通过（插值）两个控制端点并接近另外两个控制顶点。通过习题9.9，可以知道为什么等式(9-23)中的常数为3。**Bézier几何矩阵**G_B由四个点构成，即 333~336

$$G_B = [P_1 \quad P_2 \quad P_3 \quad P_4] \tag{9-24}$$

下面的等式是式(9-24)的矩阵形式，其中的4 × 4矩阵就是定义Hermite几何矩阵G_H与Bézier几何矩阵G_B之间关系式$G_H = G_B \cdot M_{HB}$的矩阵M_{HB}：

$$G_H = [P_1 \ \ P_4 \ \ R_1 \ \ R_4] = [P_1 \ \ P_2 \ \ P_3 \ \ P_4]\begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} = G_B \cdot M_{HB} \tag{9-25}$$

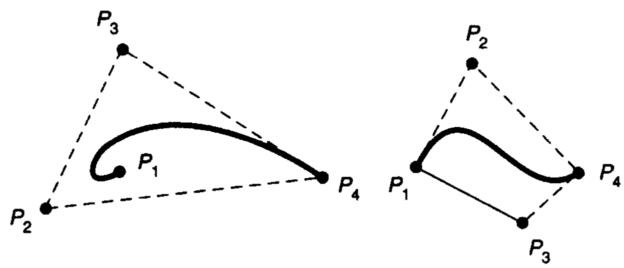


图9-15 两条Bézier曲线及其控制点。注意，虚线表示的控制顶点的凸包并不一定要经过所有四个控制点

为了求出Bézier基矩阵M_B，我们将$G_H = G_B \cdot M_{HB}$代入式(9-20)，并定义$M_B = M_{HB} \cdot M_H$得：

$$Q(t) = G_H \cdot M_H \cdot T = (G_B \cdot M_{HB}) \cdot M_H \cdot T = G_B \cdot (M_{HB} \cdot M_H) \cdot T = G_B \cdot M_B \cdot T \tag{9-26}$$

计算乘法$M_B = M_{HB} \cdot M_H$，得

$$M_B = M_{HB} \cdot M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{9-27}$$

则乘积$Q(t) = G_B \cdot M_B \cdot T$为

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \tag{9-28}$$

式(9-28)中作为权因子的四个系数多项式$B_B = M_B \cdot T$称为Bernstein多项式，如图9-16所示。

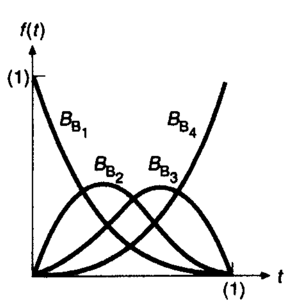


图9-16 Bézier曲线的权函数，Bernstein多项式。当$t = 0$时，只有B_{B_1}非零，因而曲线插值P_1；类似地，$t = 1$时，只有B_{B_4}非零，所以曲线插值P_4

1. 曲线段的连接

图9-17给出两条有公共端点的Bézier曲线段。当$P_3 - P_4 = k(P_4 - P_5)$，$k>0$时，端点处是G^1连续的。也就是说，P_3，P_4和P_5三点相异且共线。更严格的情况是，当$k = 1$，G^1连续变成了C^1连续。

如果把两段曲线的多项式表示为x^l（左段）和x^r（右段），就可以写出拼接点处具有C^0和C^1的连续性的条件：

$$x^l(1) = x^r(0), \quad \frac{d}{dt}x^l(1) = \frac{d}{dt}x^r(0) \tag{9-29}$$

只考虑等式(9-29)中的x分量，得到

$$x^l(1) = x^r(0) = P_{4_x}, \frac{d}{dt}x^l(1) = 3(P_{4_x} - P_{3_x}), \frac{d}{dt}x^r(0) = 3(P_{5_x} - P_{4_x}) \tag{9-30}$$

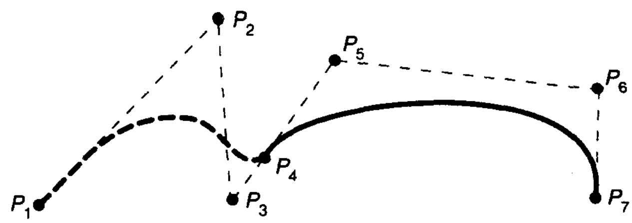


图9-17 两条Bézier曲线的拼接点是P_4。顶点P_3，P_4，P_5共线

同样，相同的条件对y和z也成立。这样，正如预料的那样，当$P_4 - P_3 = P_5 - P_4$时，曲线具有C^0和C^1连续性。

2. 凸包的重要性

考察式(9-28)和图9-16中的四个多项式B_B，可知，在$0 \leq t < 1$内每一点，它们的和总为1，而且每一个多项式值都为非负。因此，$Q(t)$就是四个控制顶点的加权平均。这说明每条曲线段都是四个控制顶点分别乘上多项式权值后的和，从而完全落在四个控制顶点的**凸包**之内。平面曲线的凸包就是四个控制顶点组成的凸多边形：将这个多边形想像成围绕在四个点上的橡皮筋（图9-15）。对于三维曲线，凸包即为控制顶点组成的多面体。将它想像为紧紧包住四个点的一张橡皮膜。

如果调配函数非负且和为1，则对所有通过控制顶点做加权和定义的三次曲线，凸包性都成立。一般来说，n个点的加权平均落在这n个点的凸包内；这对$n = 2$或$n = 3$显然成立，对更大的n可归纳证明。由四个多项式和为1这一性质，可得另一个推论：对任意的t，第四个多项式的值等于1减去前三个多项式的值。

凸包性对曲线裁剪也十分有用。我们不用对曲线中每一条直线段进行裁剪以决定其是否可见。首先对曲线凸包或它的包围区域应用多边形裁剪算法（例如，第3章中讨论的Sutherland-Hodyman算法）。若凸包（包围区域）完全在裁剪区域内，则整个曲线段也在区域内；若凸包（包围区域）完全在裁剪区域外，则曲线段也在区域外；只有当凸包（包围区域）裁剪区相交时才要对曲线本身进行裁剪。 337~338

例9.2

问题：用SRGP写一个程序，允许用户指定Bézier曲线的4个控制点，并用程序9.1中的方法画出该曲线。应该提供一个方法来规定任意数目的Bézier曲线，清除SRGP窗口，以及终止程序。

解答：我们用式(9-28)来实现DrawCurve 函数，它将曲线$Q(t)$与4个控制点关联在一起。通常，这个实现为了清晰而牺牲效率。然而，我们还是用一个最有效的方法——SRGP_polyLine函数来画曲线。该实现的余下部分参照程序9.1的模型。

我们任意指定窗口的大小和逼近曲线的步数，分别为400和20。有许多可能的方法来实现程序的交互部分。我们选择使用定位器和键盘设备的组合。定位器的右键用来指定一个新控制点序列的开始，而左键用来定义期于三个点。橡皮筋线回应有助于引导安排点的布局。一旦输入了最后一个点，Bézier曲线就被画出。

最后，用户按c键，窗口即清除；按q键，程序终止。由这个程序生成的一组典型的曲线展示在相应的图中。

交互生成Bézier曲线的程序如下：

```
#include "srgp.h"
#include <stdio.h>

#define KEYMEASURE_SIZE    80
#define WINDOW_SIZE        400
#define NUM_STEPS          20

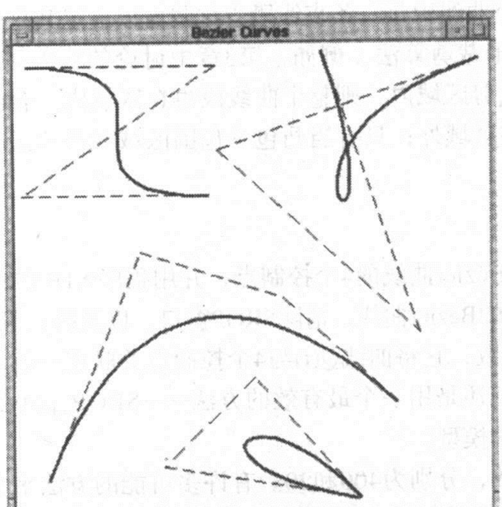
void  DrawCurve(point *ControlPoints, int n)
{
   int    i;
   float  t, delta;
   point  CurvePoints[n];

   CurvePoints[0].x = ControlPoints[0].x;      /* Bézier曲线插值的第一个控制点 */
   CurvePoints[0].y = ControlPoints[0].y;      /* 及最后一个控制点 */
   delta = 1.0 / n;                       /* 曲线用n个点来逼近 */
                                          /* t的范围为0.0到1.0 */

   for (i = 1; i <= n; i++) {
      t = i * delta;
      CurvePoints[i].x = ControlPoints[0].x * (1.0 – t) * (1.0 – t) * (1.0 – t)
          + ControlPoints[1].x * 3.0 * t * (1.0 – t) * (1.0 – t)
          + ControlPoints[2].x * 3.0 * t * t * (1.0 – t)
          + ControlPoints[3].x * t * t * t;

      CurvePoints[i].y = ControlPoints[0].y * (1.0 – t) * (1.0 – t) * (1.0 – t)
          + ControlPoints[1].y * 3.0 * t * (1.0 – t) * (1.0 – t)
          + ControlPoints[2].y * 3.0 * t * t * (1.0 – t)
          + ControlPoints[3].y * t * t * t;
   }
   SRGP_polyLine(n + 1, CurvePoints);          /* 画完整的曲线 */
}
```

339



Bézier曲线程序的典型输出

```
main()
{
   locator_measure locMeasure, pastlocMeasure;
   char   keyMeasure[KEYMEASURE_SIZE];
   int    device;
   int    numCtl;
   boolean terminate;
```

```
    rectangle screen;
    point  ControlPoints[4];

    SRGP_begin("Bezier Curves", WINDOW_SIZE, WINDOW_SIZE, 1, FALSE);
    SRGP_setLocatorEchoType(CURSOR);
    SRGP_setLocatorButtonMask(LEFT_BUTTON_MASK|RIGHT_BUTTON_MASK);
    pastlocMeasure.position = SRGP_defPoint(-1, -1);    /* 将位置初始化到 */
    SRGP_setLocatorMeasure(pastlocMeasure.position);   /* 任意位置 */
    SRGP_setKeyboardProcessingMode(RAW);
    SRGP_setInputMode(LOCATOR, EVENT);            /* 定位器（鼠标） */
    SRGP_setInputMode(KEYBOARD, EVENT);           /* 和键盘 */
    screen = SRGP_defRectangle(0, 0, WINDOW_SIZE - 1, WINDOW_SIZE - 1);

/* 主事件循环 */
    terminate = FALSE;
    do {
      device = SRGP_waitEvent(INDEFINITE);
      switch (device) {
      case KEYBOARD:{
            SRGP_getKeyboard(keyMeasure, KEYMEASURE_SIZE);
            switch (keyMeasure[0]) {
            case 'q':                    /* 退出程序 */
               terminate = TRUE;
               break;
            case 'c':                    /* 清除窗口 */
               SRGP_setColor(0);
               SRGP_fillRectangle(screen);
               SRGP_setColor(1);
               break;
            }
            break;
         }                               /* 键盘的情况 */
      case LOCATOR:{
            SRGP_getLocator(&locMeasure);
            switch (locMeasure.buttonOfMostRecentTransition) {
            case LEFT_BUTTON:          /* 定义余下的控制点 */
               if ((locMeasure.buttonChord[LEFT_BUTTON] == DOWN) &&
                  pastlocMeasure.position.x > 0) {
                  SRGP_setLocatorEchoRubberAnchor(locMeasure.position);
                  SRGP_line(pastlocMeasure.position, locMeasure.position);
                  pastlocMeasure = locMeasure;
                  ControlPoints[numCtl] = locMeasure.position;
                  numCtl++;
                  if (numCtl == 4) {
                     SRGP_setLineStyle(CONTINUOUS);      /* 画曲线 */
                     SRGP_setLineWidth(2);
                     DrawCurve(ControlPoints, NUM_STEPS);
                     pastlocMeasure.position.x = -1;
                     SRGP_setLocatorEchoType(CURSOR);
                  }
                  break;
            case RIGHT_BUTTON:         /* 开始新一组的控制点 */
                  SRGP_setLocatorEchoRubberAnchor(locMeasure.position);
                  pastlocMeasure = locMeasure;
                  SRGP_setLocatorEchoType(RUBBER_LINE);
                  SRGP_setLineStyle(DASHED);     /* 画控制多边形 */
                  SRGP_setLineWidth(1);
                  ControlPoints[0] = locMeasure.position;
                  numCtl = 1;
                  break;
```

340

341

```
                }
            }                   /* 按键处理 */
        }                       /* 定位器的情况 */
    }                           /* device switch */
} while (!terminate);
SRGP_end();
}
```

9.2.4 均匀非有理B样条曲线

样条这个词可追溯到绘图人员在设计飞机、汽车或轮船的表面时使用的有弹性的金属长条。通过移动压在样条上的“压铁”，可以沿任意方向推动样条。除非受到太大的压力，否则这些金属样条具有二阶连续性。这些金属样条的数学描述称为**自然三次样条**，它具有C^0，C^1和C^2连续性并插值控制点。这种曲线的连续性比Hermite曲线和Bézier曲线高一次，从而看起来也更加光滑。

但是，自然三次样条的多项式系数由所有的n个控制点决定，并需要对一个$(n+1)\times(n+1)$的矩阵求逆[BART87]。这就带来了两个缺点：一个控制点的改变将影响整条曲线，对矩阵求逆的计算时间会影响曲线交互调整的速度。

本节讨论的**B样条**由若干条曲线段构成，每个曲线段的多项式系数仅由少数几个控制点决定，称之为**局部控制**。这样，某个控制顶点的改变只影响曲线的一小部分。同时，计算系数的时间也大大缩短了。B样条具有和自然样条同样的连续性，但不插值控制顶点。

在后面的讨论中，术语会有一点变化，因为我们要讨论的是由几个曲线段构成的一整条曲线，而不是单独的一条曲线段。一条曲线段不需要经过控制点，而且它的两个连续性条件要从相邻的曲线段获得。为此曲线段之间要共用控制点，所以最好把这些曲线段看成一个整体来描述。

三次B样条曲线由$m-2$条三次多项式曲线段$Q_3, Q_4, \cdots, Q_m$构成的曲线来逼近$m+1$个控制点$P_0, P_1, \cdots, P_m$, $m\geqslant 3$。每一条三次曲线段的参数域是 $0\leqslant t<1$，但通过调整参数域（用$t=t+k$代入）使不同曲线段的参数域相接。这样，Q_i的参数域为$t_i\leqslant t<t_{i+1}$，$3\leqslant i\leqslant m$，特殊情况$m=3$时，曲线段Q_3的四个控制点是P_0到P_3，参数域为区间$t_3\leqslant t<t_4$。

对每一个$i\geqslant 4$，在参数值t_i处曲线段Q_{i-1}和Q_i之间有一个拼接点或叫**结点**；这种点的参数值
342 称为**结点值**。因为t_3与t_{m+1}所定义的始点与终点也是结点，所以总共有$m-1$个结点。图9-18给出了标明结点的平面B样条曲线。容易构造一条闭合的B样条曲线：控制点P_0，P_1和P_2在控制点序列末尾重复一次，即$P_0, P_1, \cdots, P_m, P_0, P_1, P_2$。

这里**均匀**的意思是结点参数t呈相同间隔分布。不失一般性，可以假设$t_3=0$且区间$t_{i+1}-t_i=1$。在9.2.5节将讨论非均匀非有理B样条，它的结点分布是不均匀的。（实际上，本节介绍的结点的概念主要是为非均匀样条做铺垫的。）**非有理**是为了区别从三次有理多项式曲线中得到的样条曲线，其中的$x(t)$，$y(t)$和$z(t)$都定义为两个三次多项式之比，在9.2.6节将讨论这种曲线。“B”代表basis，因为这种样条曲线可以表示成多项式基函数加权和的形式，而自然样条则不可以。

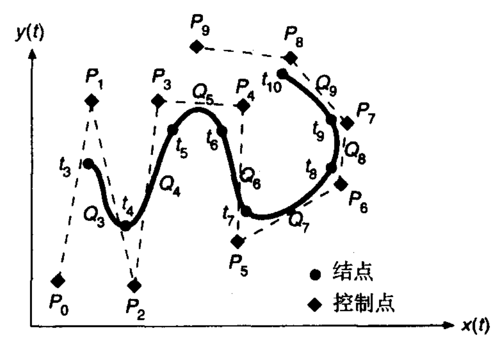


图9-18 由曲线段Q_3到Q_9构成的B样条。此图以及本章中其他许多图的生成程序是由Carles Castellsaquè所写

B样条曲线的$m+1$个控制点定义了$m-2$条曲线段，其中的每一条曲线段都由四个控制点定义。特别是，曲线段Q_i由点P_{i-3}，P_{i-2}，P_{i-1}和P_i定义。因此，曲线段Q_i的**B样条几何矩阵**G_{Bs_i}为

$$G_{Bs_i}=[P_{i-3}\ \ P_{i-2}\ \ P_{i-1}\ \ P_i],\quad 3\leqslant i\leqslant m \tag{9-31}$$

第一条曲线段Q_3由P_0到P_3定义，其参数域从$t_3=0$到$t_4=1$；Q_4由P_1到P_4定义，参数域从$t_4=1$到$t_5=2$；而最后一条曲线段Q_m由P_{m-3}, P_{m-2}, P_{m-1}和P_m定义，参数域从$t_m=m-3$到$t_{m+1}=m-2$。一般来说，曲线段Q_i从点P_{i-2}附近为起点，在点P_{i-1}附近终止。可以证明B样条的调配函数处处非负而且和为1，所以曲线段Q_i限制在它的四个控制点所构成的凸包之内。 343

正如每条曲线段由四个控制点定义一样，每一个控制点（除控制点序列P_0，P_1，…，P_m的起始点和终止点外）能影响到四条曲线段。沿给定方向移动一个控制点，则受此点影响的四条曲线段都会沿相同方向移动；而其余曲线段则完全不受影响（如图9-19所示）。这就是B样条的局部控制性，本章所介绍的所有样条曲线也都有这样的性质。

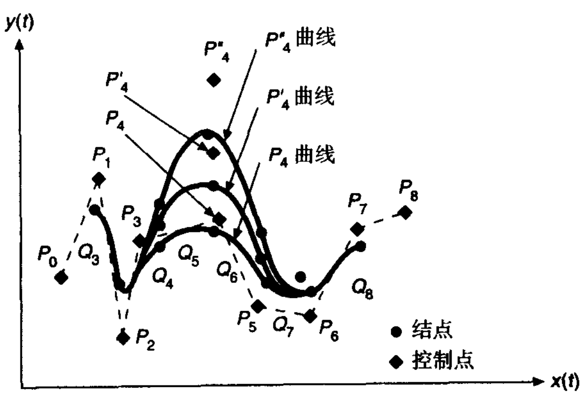


图9-19 当控制顶点P_4在不同位置时B样条曲线的形状

若将T_i定义为行向量$[(t-t_i)^3\ (t-t_i)^2\ (t-t_i)1]^{\mathrm{T}}$，则第$i$条曲线段的B样条公式为

$$Q_i(t)=G_{Bs_i}\cdot M_{Bs}\cdot T_i,\quad t_i\leqslant t<t_{i+1} \tag{9-32}$$

对$3\leqslant i\leqslant m$，应用式(9-32)就可得到整条曲线。

B样条基矩阵M_{Bs}建立了几何约束G_{Bs}与调配函数和多项式系数之间的关系：

$$M_{Bs}=\frac{1}{6}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix} \tag{9-33}$$

上述矩阵的推导见[BART87]。

与前面Bézier曲线和Hermite曲线类似，B样条的调配函数B_{Bs}可以通过乘积$M_{Bs}\cdot T_i$得到。注意，因为每条曲线段i的参数值$t-t_i$从$t=t_i$时为0变化到$t=t_{i+1}$时为1，所以每条曲线段的调配函数完全相同。若用$t-t_i$代替t，而且将区间$[t_i, t_{i+1}]$换为[0，1]，就得到 344

$$\begin{aligned}B_{Bs}=M_{Bs}\cdot T&=[B_{Bs\text{-}3}\ \ B_{Bs\text{-}2}\ \ B_{Bs\text{-}1}\ \ B_{Bs0}]^{\mathrm{T}}\\ &=\frac{1}{6}[-t^3+3t^2-3t+1\ \ \ 3t^3-6t^2+4\ \ \ -3t^3+3t^2+3t+1\ \ \ t^3]^{\mathrm{T}}\\ &=\frac{1}{6}[(1-t)^3\ \ \ 3t^3-6t^2+4\ \ \ -3t^3+3t^2+3t+1\ \ \ t^3]^{\mathrm{T}},\quad 0\leqslant t<1\end{aligned} \tag{9-34}$$

图9-20是B样条的调配函数B_{Bs}。因为四个函数都非负且总和为1，凸包性对每个曲线段都成立。由[BART87]可知这些调配函数与Bernstein多项式基函数之间的关系。

展开式(9-32)，在第二个等号后将$t-t_i$换成t，得到

$$\begin{aligned}Q_i(t-t_i) &= G_{Bs_i}\cdot M_{Bs}\cdot T_i = G_{Bs_i}\cdot M_{Bs}\cdot T\\ &= G_{Bs_i}\cdot B_{Bs} = P_{i-3}\cdot B_{Bs-3}+P_{i-2}\cdot B_{Bs-2}+P_{i-1}\cdot B_{Bs-1}+P_i\cdot B_{Bs0}\\ &= \frac{(1-t)^3}{6}P_{i-3}+\frac{3t^3-6t^2+4}{6}P_{i-2}+\frac{-3t^3+3t^2+3t+1}{6}P_{i-1}\\ &+\frac{t^3}{6}P_i,\quad 0\leqslant t<1\end{aligned} \tag{9-35}$$

容易证明Q_i和Q_{i+1}之间在拼接点处具有C^0，C^1和C^2连续性。B样条曲线可以有更强的连续性，但这是以降低控制的灵活性为代价的。通过相重的控制点，曲线可以插值某个控制点；这对曲线的端点和中间点都有用。比如，若$P_{i-2}=P_{i-1}$，曲线就会被这个控制点拉近，因为曲线段Q_i只由三个不同点定义，而$P_{i-2}=P_{i-1}$在式(9-35)中就被权因子乘了两次：一次是B_{Bs-2}，另一次是B_{Bs-1}。

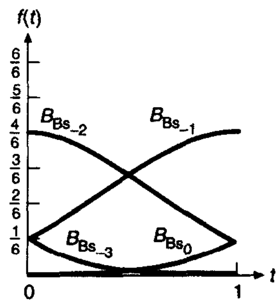


图9-20 式(9-34)的四个B样条调配函数。当$t=0$或$t=1$时，恰有三个函数非零

若一个控制点被使用三次，比如，$P_{i-2}=P_{i-1}=P_i$，则式(9-35)就变成了

$$Q_i(t)=P_{i-3}\cdot B_{Bs-3}+P_i\cdot(B_{Bs-2}+B_{Bs-1}+B_{Bs0}) \tag{9-36}$$

Q_i显然是一条直线。而且，在$t=1$时直线插值P_{i-2}，此时P_i的三个系数和为1，但一般地，$t=0$时曲线不插值P_{i-3}。也可以这样想，Q_i的凸包现在只定义在两个不同点上，所以Q_i只能是一条直线。图9-21显示了多重控制点对B样条曲线内部的影响。

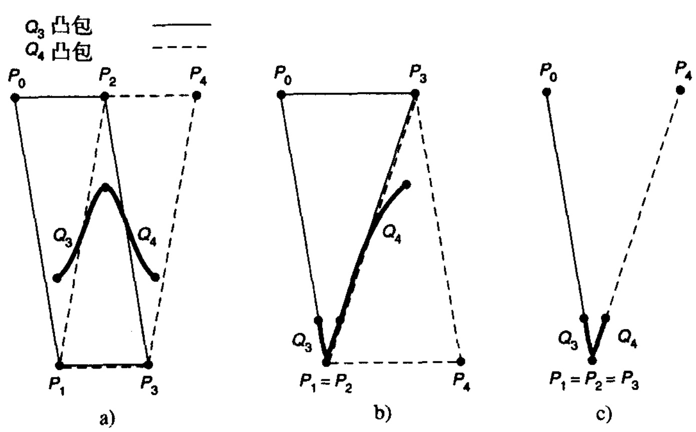


图9-21 均匀B样条曲线上多重控制点的影响。a)没有多重顶点，两曲线段的凸包有重叠；曲线段Q_3和Q_4之间的拼接点在两凸包的公共区域内。b)中有一个二重控制点，两凸包共享一条边P_2P_3，曲线段间的拼接点落在共享边上。c)有一个三重控制点，两凸包是直线段，在三重控制点处相接，因此，两曲线段之间的拼接点也是此三重控制点。因两凸包是直线段，所以，两曲线段也必须是直线段。在拼接点处是C^2参数连续的，但只是G^0几何连续

在[BARS83；BART87]中介绍了另一种插值端点（**虚顶点**）的方法。下一节我们将看到，用非均匀B样条可以比用均匀B样条更自然地使曲线插值端点或中间点。

9.2.5 非均匀非有理B样条曲线

非均匀非有理B样条曲线与9.2.4节中讨论的均匀非有理B样条曲线的区别在于相邻结点值所取的参数间隔不一定是均匀的。不均匀的结点值序列意味着每一段曲线上的调配函数不再相同。

这样的曲线比均匀B样条有几个优点。首先，选定拼接点上的连续性可以从C^2减为C^1或C^0，甚至没有连续性。若连续性减为C^0，则曲线可以插值一个控制点而不会产生像均匀B样条那样控制点两边的曲线都退化成直线的现象。其次，曲线可以恰好插值给定的始点与终点而不会出现变成直线段的曲线段。正如[FOLE90]中讨论的那样，对非均匀B样条还可以添加结点和控制点，这样很容易改变一条曲线的形状，而这对均匀B样条是做不到的。

由于非均匀B样条比均匀B样条具有更一般的特性，需要把均匀B样条中用到的表示方法稍微变化一下。与前面一样，非均匀B样条也是由三次多项式曲线构成的分段连续曲线，逼近控制点P_0到P_m。**结点值序列**是一个从t_0到t_{m+4}的单调不减序列（也就是说，结点数要比控制点数多四个）。因为控制点至少为四个，所以最小的结点序列也至少有八个结点值，而且曲线定义在从t_3到t_4的参数区间。 345~346

结点序列的惟一限制是单调不减，也就是允许相邻结点值相等。若相等，则参数值称为**重结点**，而相等的参数值的数目称为结点的**重数**（单独一个结点的重数为1）。举个例子，在结点序列（0, 0, 0, 0, 1, 1, 2, 3, 4, 4, 5, 5, 5, 5）中，结点值0重数为4；结点值1的重数为2；值2和3的重数都为1；值4的重数为2；值5的重数为4。

曲线段Q_i由控制点$P_{i-3}, P_{i-2}, P_{i-1}, P_i$和调配函数$B_{i-3,4}(t), B_{i-2,4}(t), B_{i-1,4}(t), B_{i,4}(t)$确定，可表示成加权和的形式

$$Q_i(t) = P_{i-3}\cdot B_{i-3,4}(t) + P_{i-2}\cdot B_{i-2,4}(t) + P_{i-1}\cdot B_{i-1,4}(t) + P_i\cdot B_{i,4}(t) \qquad (9\text{-}37)$$
$$3 \leqslant i \leqslant m, \quad t_i \leqslant t < t_{i+1}$$

在从t_3到t_{m+1}的区间以外，曲线没有定义。当$t_i = t_{i+1}$时（即重结点情形），曲线段Q_i是一个点。正是这种允许曲线段退化为点的表示方法，使非均匀B样条具有更多的灵活性。

与其他类型的样条曲线不同，非均匀B样条的调配函数不惟一。它们依赖于结点值之间的区间，由低阶的调配函数递归定义。$B_{i,j}(t)$是第j阶调配函数，作为控制点P_i的权因子。因为我们讨论的是四阶（三次）B样条，递归定义终止于$B_{i,4}(t)$，并且很容易表示成嵌套形式。三次B样条的递归形式如下：

$$
\begin{aligned}
B_{i,1}(t) &= \begin{cases} 1, & t_i \leqslant t < t_{i+1} \\ 0, & \text{其他} \end{cases} \\
B_{i,2}(t) &= \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t) \\
B_{i,3}(t) &= \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t) \\
B_{i,4}(t) &= \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t)
\end{aligned} \qquad (9\text{-}38)
$$

可以证明调配函数是非负且和为1的，所以非均匀B样条曲线完全在它们的四个控制点的凸包之内。对于重数大于一的结点，由于相邻结点值相等，分母有可能为零：因此规定除以零的结果为零。

增加结点的重数有两种效果。第一种，每个结点值t_i自动处于点P_{i-3}，P_{i-2}和P_{i-1}的凸包之内。若t_i和t_{i+1}相等，则它们必须同时处于P_{i-3},P_{i-2},P_{i-1}凸包和P_{i-2},P_{i-1},P_i凸包中。这就是说它们必须恰好在线段P_{i-2}, P_{i-1}上。同样，若$t_i=t_{i+1}=t_{i+2}$，则这个结点必须在点P_{i-1}上。若$t_i=t_{i+1}=$
347 $t_{i+2}=t_{i+3}$，则结点必须在P_{i-1}和P_i上——曲线就断开了。第二种，重结点会降低参数连续性：增加一个结点（重数为2），连续性从C^2降为C^1；增加两个结点（重数为3），连续性从C^1降为C^0；增加3个结点（重数为4），则曲线从C^0连续变为不连续。

典型的交互生成非均匀样条曲线方法，要求给定控制点的位置，只要连续选中同一个点就能指定一个重控制点。另一种方法是用一个多键鼠标直接点曲线：双击一个键表示一个双重控制点；双击另一个键表示一个双重结点。

9.2.6 非均匀有理三次多项式曲线段

一般的有理三次曲线段是多项式的比：

$$x(t)=\frac{X(t)}{W(t)},\quad y(t)=\frac{Y(t)}{W(t)},\quad z(t)=\frac{Z(t)}{W(t)} \tag{9-39}$$

其中$X(t)$，$Y(t)$，$Z(t)$和$W(t)$都是三次多项式曲线，它们的控制点定义在齐次坐标上。还可以将曲线想像成定义在齐次空间$Q(t)=[X(t), Y(t), Z(t), W(t)]^T$上。通常，从齐次空间变到三维空间要除以$W(t)$。任一个非有理曲线可以通过增加第四个元素$W(t)=1$变成有理曲线。一般，有理曲线中的多项式可以是Bézier、Hermite或其他形式的。如果是B样条，就得到非均匀有理B样条曲线，有时称为NURBS曲线[FORR80]。

有理曲线有两个优点。首先，也是最重要的是，它在控制点的旋转、缩放、平移以及透视变换下具有不变性（非有理曲线只在旋转、缩放、平移下有不变性）。因此，只要对控制点运用透视变换就能对原曲线进行透视变换。如果在透视变换前不想把非有理曲线先转化为有理曲线，就得对曲线的每个点做透视变换，这样效率就要低得多。类似地可以观察到，对一个球体做透视变换与只对原来球体的球心和半径做变换所得的球是不同的。

有理曲线的第二个优点是可以精确定义圆锥曲线。这一点与非有理曲线不同，圆锥曲线可以用非有理曲线逼近，但需要用许多接近圆锥曲线的控制点。这个优点在一些应用领域，特别是CAD领域，非常有用。在那里，同时需要一般的曲线、曲面以及圆锥曲线，两种实体类型都能用NURBS曲线表示。

圆锥曲线和NURBS曲线的进一步讨论，可以参阅[FAUX79；BÖHM84；TILL83]。

9.2.7 用数字化点来拟合曲线

工程师和艺术家常常会有一个复杂形状的非电子化表示，该形状是可以用一组离散点来予
348 以数字化的。例如，形状的纸质硬拷贝可以全部利用起来。对该形状进行附加的操作，我们可以用一条或一组平滑的曲线来构造该形状的（通常是）非精确的数字化表示。已经发表了许多不同的曲线拟合技术，它们有不同的优点和缺点。Schneider [SCHN90]开发了用分段Bézier曲线来逼近数字化曲线的方法。该方法的优点是几何连续性、稳定性及易于实现。该算法的一个完整的C程序实现，请见[SCHN90]。图9-22是该方法应用于一个数字化形状的例子。

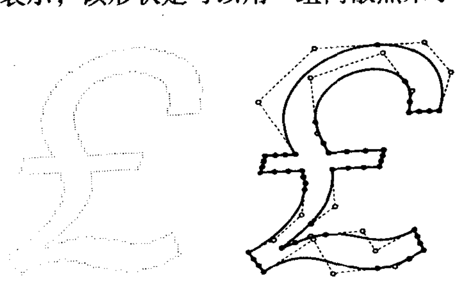


图9-22 显示了原有样品、拟合后曲线及Bézier曲线段控制点的一个数字化字符。（Academic Press公司授权。）

9.2.8 三次曲线的比较

不同类型的三次参数曲线可以从几个不同方面进行比较，比如交互控制的灵活性、在拼接点处的连续程度、表示的一般性和计算速度等。当然，既然所有表示都可以相互转化（如[FOLE90]中讨论的那样），没有必要只选择单独一种表示法。举个例子，可以用非均匀有理B样条作为内部表示，而用户可以交互控制Bézier控制点或者Hermite控制点和切向量。一些交互的图形编辑系统为用户提供Hermite曲线，同时在内部表示时为PostScript支持的Bézier形式[ADOB85]。一般来说，一个交互CAD系统的用户可能有几种选择，如Hermite、Bézier、均匀B样条和非均匀B样条等。因为非均匀有理B样条最具一般性，它经常用于系统的内部表示。

表9-1对本节提到过的大多数曲线进行了比较。表中并没有直接给出交互控制灵活性的比较，因为它更依赖于具体的应用领域。*控制曲线段的参数数量*是指四个几何条件以及其他参数，比如非均匀样条曲线的结点分布。*容易达到的连续性*指一些约束条件，比如使控制点共线以达到G^1连续性等。因为C^n连续性要比G^n强，任何一种能达到C^n连续的曲线理论上也能达到G^n。 349

表9-1 七种三次参数曲线的比较

	Hermite	Bézier	均匀B样条	非均匀B样条
控制点定义的凸包	N/A	Yes	Yes	Yes
插值一些控制点	Yes	Yes	No	No
插值所有控制点	Yes	No	No	No
易于分割	Cood	Best	Avg	High
表示法固有的连续性	C^0	C^0	C^2	C^2
	G^0	G^0	G^2	G^2
容易达到的连续性	C^1	C^1	C^2	C^2
	G^1	G^1	$G^2$①	$G^2$①
控制一曲线段的参数个数	4	4	4	5

① 在9.2节讨论的特殊情况除外。

CAD系统经常只要求几何连续性，此时，选择范围就缩小到几种样条曲线上，它们都能达到G^1和G^2连续。在表中的三种样条曲线中，均匀B样条曲线限制最多。非均匀B样条所允许的多重结点，使得用户对曲线形状的控制更灵活。当然，重要的是要有一个好的用户界面，使用户能充分利用这些功能。

传统上用户应该能交互地拖动控制点或切向量，并及时更新曲线显示。图9-19显示了这样一个B样条序列。在一些应用中，B样条曲线的一个缺点是控制点不在曲线上。但也可以不显示控制点，而让用户交互控制结点（做上标记以便选取）。 350

9.3 双三次参数曲面

双三次参数曲面是三次参数曲线的推广。回忆一下三次参数曲线的一般形式$Q(t) = G \cdot M \cdot T$，其中几何矩阵G是一个常量。首先，为了定义方便，我们将t换成s，得到$Q(s) = G \cdot M \cdot S$。如果允许G中的点可以在三维空间中沿某条以t为参数的路径变化，则得到

$$Q(s,t) = [G_1(t) \;\; G_2(t) \;\; G_3(t) \;\; G_4(t)] \cdot M \cdot S \tag{9-40}$$

现在，对一个固定的t_1，$Q(s, t_1)$是一条曲线，因为$G(t_1)$是常量。让t变到另一个值，比如，t_2，其中$t_2 - t_1$很小，则$Q(s, t_2)$略有不同。令t_2在0和1之间重复取值，就定义了整个曲线网格，相邻

两条曲线都可任意接近。所有这些曲线的集合就定义了一个曲面。若$G_i(t)$本身是三次曲线，则这样的曲面称为**双三次参数曲面**。

考虑$G_i(t)$为三次曲线的情形，每一个都可以表示为$G_i(t) = \boldsymbol{G}_i \cdot \boldsymbol{M} \cdot \boldsymbol{T}$，其中$\boldsymbol{G}_i = [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]$（$\boldsymbol{G}$和$g$用来区别曲线中用到的$G$）。因此，$g_{i1}$就是曲线$G_i(t)$的几何矩阵中的第一个元素，以此类推。

利用恒等式$(A \cdot B \cdot C)^{\mathrm{T}} = C^{\mathrm{T}} \cdot B^{\mathrm{T}} \cdot A^{\mathrm{T}}$，对式$G_i(t) = \boldsymbol{G}_i \cdot \boldsymbol{M} \cdot \boldsymbol{T}$做转置，得到$G_i(t) = \boldsymbol{T}^{\mathrm{T}} \cdot \boldsymbol{M}^{\mathrm{T}} \cdot \boldsymbol{G}_i^{\mathrm{T}} = \boldsymbol{T}^{\mathrm{T}} \cdot \boldsymbol{M}^{\mathrm{T}} \cdot [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]$。将这个结果代入式(9-40)，得到

$$Q(s,t) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot \begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix} \cdot M \cdot S \tag{9-41}$$

或者

$$Q(s,t) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot \boldsymbol{G} \cdot M \cdot S, \quad 0 \leqslant s, t \leqslant 1 \tag{9-42}$$

把x，y，z分开表示，则形式为

$$x(s,t) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot \boldsymbol{G}_x \cdot M \cdot S$$

$$y(s,t) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot \boldsymbol{G}_y \cdot M \cdot S$$

$$z(s,t) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot \boldsymbol{G}_z \cdot M \cdot S \tag{9-43}$$

给定这样的一般形式，现在可以继续探讨用不同的几何矩阵描述特定曲面的特定方法了。

9.3.1 Hermite曲面

351 Hermite曲面完全由一个4×4的几何矩阵$\boldsymbol{G}_{\mathrm{H}}$定义。使用与式(9-42)同样的推导方法，可以求出$\boldsymbol{G}_{\mathrm{H}}$。这里，再把推导过程深入展开一些，代入$x(s, t)$。首先，把式(9-13)中的$t$换成$s$，得到$x(s) = \boldsymbol{G}_{\mathrm{H}_x} \cdot \boldsymbol{M}_{\mathrm{H}} \cdot \boldsymbol{S}$。重新改写这个等式，使$\boldsymbol{G}_{\mathrm{H}x}$不是常量，而是关于$t$的函数，得到

$$x(s,t) = G_{\mathrm{H}_x}(t) \cdot M_{\mathrm{H}} \cdot S = [P_{1_x}(t)\ \ P_{4_x}(t)\ \ R_{1_x}(t)\ \ R_{4_x}(t)] \cdot M_{\mathrm{H}} \cdot S \tag{9-44}$$

函数$P_{1_x}(t)$和$P_{4_x}(t)$定义了关于s的参数曲线的初始点和终止点的x分量。类似地，$R_{1_x}(t)$和$R_{4_x}(t)$是这两点上的切向量。对于任一给定的t值，可随之确定两端点和两切向量。图9-23给出了$P_1(t)$，$P_4(t)$以及当 t 等于0.0, 0.2, 0.4, 0.6, 0.8和1.0时关于s的三次曲线。实际上也可以把这块曲面看成$P_1(t) = Q(0, t)$和$P_4(t) = Q(1, t)$之间的三次插值，或者是$Q(s, 0)$和$Q(s, 1)$之间的三次插值。

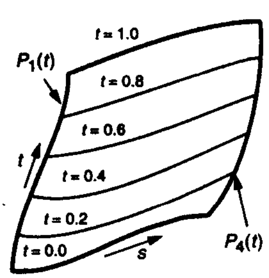


图9-23 双三次曲面上的等参线：$P_1(t)$取$s = 0$，$P_4(t)$取$s = 1$

特殊情况是，四条插值曲线$Q(0, t)$，$Q(1, t)$，$Q(s, 0)$和$Q(s, 1)$都是直线，得到的是一个**直纹面**。若插值曲线共面，则曲面就成了一个平面四边形。

继续推导，把$P_{1_x}(t), P_{4_x}(t), R_{1_x}(t)$和$R_{4_x}(t)$都表示为Hermite形式

$$P_{1_x}(t) = [g_{11}\ \ g_{12}\ \ g_{13}\ \ g_{14}]_x \cdot M_{\mathrm{H}} \cdot T, \quad P_{4_x}(t) = [g_{21}\ \ g_{22}\ \ g_{23}\ \ g_{24}]_x \cdot M_{\mathrm{H}} \cdot T$$

$$R_{1_x}(t) = [g_{31}\ \ g_{32}\ \ g_{33}\ \ g_{34}]_x \cdot M_{\mathrm{H}} \cdot T, \quad R_{4_x}(t) = [g_{41}\ \ g_{42}\ \ g_{43}\ \ g_{44}]_x \cdot M_{\mathrm{H}} \cdot T \tag{9-45}$$

这四条三次曲线可以合为一个等式：

$$[P_{1_x}(t)\ \ P_{4_x}(t)\ \ R_{1_x}(t)\ \ R_{4_x}(t)]^{\mathrm{T}} = G_{\mathrm{H}_x} \cdot M_{\mathrm{H}} \cdot T \tag{9-46}$$

其中

$$G_{\mathrm{H}_x}=\begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \tag{9-47}$$

将式(9-46)两边一起作转置，得

$$[P_{1_x}(t)\ \ P_{4_x}(t)\ \ R_{1_x}(t)\ \ R_{4_x}(t)] = T^{\mathrm{T}}\cdot M_{\mathrm{H}}^{\mathrm{T}}\cdot\begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix}_x = T^{\mathrm{T}}\cdot M_{\mathrm{H}}^{\mathrm{T}}\cdot G_{\mathrm{H}_x} \tag{9-48}$$

将式(9-48)代入式(9-44)得

$$x(s,t)=T^{\mathrm{T}}\cdot {M_{\mathrm{H}}}^{\mathrm{T}}\cdot G_{\mathrm{H}_x}\cdot M_{\mathrm{H}}\cdot S \tag{9-49}$$

类似地，

$$y(s,t)=T^{\mathrm{T}}\cdot {M_{\mathrm{H}}}^{\mathrm{T}}\cdot G_{\mathrm{H}_y}\cdot M_{\mathrm{H}}\cdot S,\quad z(s,t)=T^{\mathrm{T}}\cdot {M_{\mathrm{H}}}^{\mathrm{T}}\cdot G_{\mathrm{H}_z}\cdot M_{\mathrm{H}}\cdot S \tag{9-50}$$

三个4×4矩阵G_{H_x}，G_{H_y}和G_{H_z}在Hermite曲面中的作用与G_{H}在Hermite曲线中的作用一样。 352

正如三次Hermite曲线在两条曲线段之间具有C^1和G^1连续性一样，Hermite双三次曲面在两个曲面片之间同样具有C^1和G^1连续性。详细内容在[FOLE90]的第11章中介绍。

9.3.2 Bézier曲面

Bézier双三次曲面的公式推导过程与Hermite三次曲线的完全一样。其结果是：

$$\begin{aligned} x(s,t)&=T^{\mathrm{T}}\cdot {M_{\mathrm{B}}}^{\mathrm{T}}\cdot G_{\mathrm{B}_x}\cdot M_{\mathrm{B}}\cdot S \\ y(s,t)&=T^{\mathrm{T}}\cdot {M_{\mathrm{B}}}^{\mathrm{T}}\cdot G_{\mathrm{B}_y}\cdot M_{\mathrm{B}}\cdot S \\ z(s,t)&=T^{\mathrm{T}}\cdot {M_{\mathrm{B}}}^{\mathrm{T}}\cdot G_{\mathrm{B}_z}\cdot M_{\mathrm{B}}\cdot S \end{aligned} \tag{9-51}$$

Bézier几何矩阵G由16个控制点构成，如图9-24所示。和Bézier曲线一样，Bézier曲面在交互设计时十分常用。一部分控制点落在曲面上，提供了精确的控制，同时还可以直接控制切向量。当Bézier曲面用于内部表示时，常用到它的凸包性和易分割性。

使四个公共控制点重合就能保证曲面片拼接边的C^0和G^0连续性。当拼接边两侧相对应的四个控制点与边界上的四个控制点的连线共线时，曲面具有G^1连续性。在图9-25中，以下几组控制点连线平共线，而且定义了4条线段，其长度有相同比率k：(P_{13}，P_{14}，P_{15})，(P_{23}，P_{24}，P_{25})，(P_{33}，P_{34}，P_{35})和(P_{43}，P_{44}，P_{45})。图9-1中的茶壶是用32个Bézier曲面片建模的，所有的拼接都确保了G^1连续性。

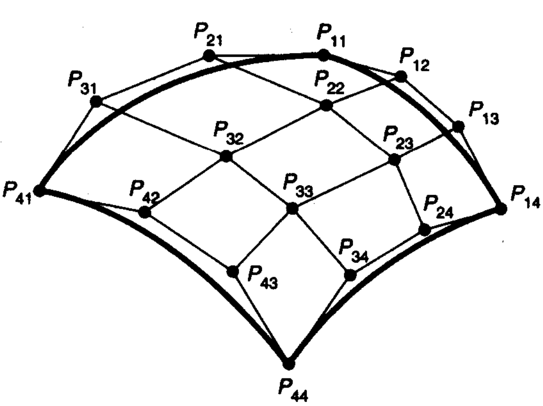


图9-24　Bézier双三次曲面片的16个控制点 353

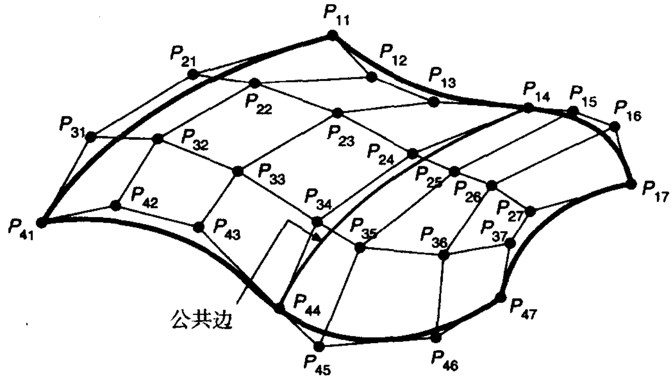


图9-25 两块在边$P_{14}, P_{24}, P_{34}, P_{44}$相拼接的Bézier曲面片

9.3.3 B样条曲面

B样条曲面片表示为

$$
\begin{aligned}
x(s,t) &= T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}{}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_x} \cdot M_{\mathrm{Bs}} \cdot S \\
y(s,t) &= T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}{}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_y} \cdot M_{\mathrm{Bs}} \cdot S \\
z(s,t) &= T^{\mathrm{T}} \cdot M_{\mathrm{Bs}}{}^{\mathrm{T}} \cdot G_{\mathrm{Bs}_z} \cdot M_{\mathrm{Bs}} \cdot S
\end{aligned}
\tag{9-52}
$$

B样条曲面在边界处自动具有C^2连续性；对控制点没有什么特殊要求，只是要避免重控制点。因为重控制点会造成不连续。

双三次非均匀有理B样条曲面以及其他有理曲面与它们的三次曲线形式类似。相应的分割和显示方法都可以直接用在双三次曲面中。

9.3.4 曲面的法线

在曲面明暗处理（第14章）、机器人碰撞检测、数控加工中的等距线计算以及其他一些计算中都需要用到双三次曲面的法线，求双三次曲面的法线较为容易。由式(9-42)可得曲面$Q(s,t)$的s切向量

$$
\begin{aligned}
\frac{\partial}{\partial s} Q(s,t) &= \frac{\partial}{\partial s}(T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S) = T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot \frac{\partial}{\partial s}(S) \\
&= T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot [3s^2 \;\; 2s \;\; 1 \;\; 0]^{\mathrm{T}}
\end{aligned}
\tag{9-53}
$$

354 以及t的切向量

$$
\begin{aligned}
\frac{\partial}{\partial t} Q(s,t) &= \frac{\partial}{\partial t}(T^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S) = \frac{\partial}{\partial t}(T^{\mathrm{T}}) \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S \\
&= [3t^2 \;\; 2t \;\; 1 \;\; 0]^{\mathrm{T}} \cdot M^{\mathrm{T}} \cdot G \cdot M \cdot S
\end{aligned}
\tag{9-54}
$$

这两个切向量在点(s,t)处与曲面平行，因此，它们的叉乘垂直于曲面。当两个向量都为零时，叉乘积也为零，这时平面法线没有意义。我们曾经说过当切向量为零时拼接点处C^1连续但不是G^1连续。

因为式(9-42)表示双三次曲面的x，y和z分量，所以每个切向量都是三元组。用x_s表示s切向量的x分量，y_s表示y分量，z_s表示z分量，则法线为

$$
\frac{\partial}{\partial s} Q(s,t) \times \frac{\partial}{\partial t} Q(s,t) = [y_s z_t - y_t z_s \quad z_s x_t - z_t x_s \quad x_s y_t - x_t y_s]
\tag{9-55}
$$

曲面的法线是一个双五次（二元，五次）多项式，因此计算起来十分麻烦。[SCHW82]中

给出了一种当曲面本身相对平滑时较为满意的双三次逼近法。

9.3.5 双三次曲面的显示

与曲线类似，曲面也可以用双三次多项式迭代求值显示。迭代求值法最适合显示图9-26的那种双三次曲面片类型。曲面上关于参数s和t的每一条等参数曲线都是三次曲线，于是可以直接显示这些曲线，如程序9-2所示。

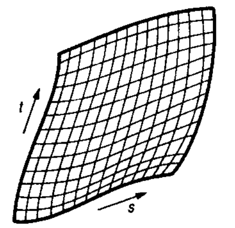


图9-26 用s和t的等参数曲线显示曲面片

对曲面直接迭代求值通常比曲线情形开销大，因为曲面方程大约要计算$2/\delta^2$次。当$\delta = 0.1$时，这个值为200；当$\delta = 0.01$时，这个值为20 000。从这些数字可以看出，另一种向前差分法更合适。这种方法和其他对显示双三次曲面有用的方法在[FOLE90; FORR79]中阐述。

程序9-2 双三次曲面片的网格显示函数。X(s, t), Y(s, t)和Z(s, t)函数分别用系数矩阵coefficients计算曲面

```
typedef float Coeffs[4][4][3];

void DrawSurface( Coeffs coefficients, int ns, int nt, int n )
   /* 变量Coefficients是Q(s, t)的系数，  */
   /* ns和nt是t和s为常数的等参数曲线被画的次数 */
{
   float del, dels, delt, s, t;
   int i, j;
   /* Initialize */
   del = 1.0 / n;              /* 每条曲线时使用的步长 */
   dels = 1.0 / (ns – 1);      /* t 为常数的等参数曲线在s方向上的步长的增量 */
   delt = 1.0 / (nt – 1);      /* s 为常数的等参数曲线在t方向上的步长的增量 */
   /* 画s为常数的等参数曲线，s = 0.0, dels, 2dels, ⋯, 1.0 */
   for ( i = 0; i < ns; i++ ) {
      s = i * dels;
      /* 画s为常数的等参数曲线，t 取值为0.0~1.0 */
      /* X, Y, Z为给定s和t的双三次曲线的求值函数 */
      MoveAbs3(X(s, 0.0), Y(s, 0.0), Z(s, 0.0));
      for ( j = 0; j < n; j++ ) {
         t = i * del;
         /* 对于每条曲线，t从0.0到1.0之间，取n步 */
         LineAbs3(X(s, t), Y(s, t), Z(s, t));
      }
   }

   /* 画t为常数的等参数曲线，t = 0.0, delt, 2delt, ⋯, 1.0 */
   for ( i = 0; i < nt; i++ ) {
      t = i * delt;
      /* 画t为常数的等参数曲线，s 取值为0.0~1.0 */
      MoveAbs3(X(0.0, t), Y(0.0, t), Z(0.0, t));
      for ( j = 0; j < n; j++ ) {
         s = j * del;
         /* 对于每条曲线，s从0到1之间，取n步 */
         LineAbs3(X(s, t), Y(s, t), Z(s, t));
      }
   }
}
```

355

对于指定类型的曲面，容易开发$X(s, t)$, $Y(s, t)$, $Z(s, t)$。作为例子，我们将考虑Bézier曲面。只要将式(9-51)中矩阵$T^T \cdot M_B^T$及$M_B \cdot S$乘开，就能改写$x(s, t)$算式为：

$$x(s,t)=[(1-t)^3 \quad 3t(1-t)^2 \quad 3t^2(1-t) \quad t^3]\cdot G_{B_x}\cdot\begin{bmatrix}(1-s)^3\\3s(1-s)^2\\3s^2(1-s)\\s^3\end{bmatrix} \tag{9-56}$$

G_{B_x}是控制点x分量的矩阵，如图9-24所示，可写成

$$G_{B_x}=\begin{bmatrix}P_{11} & P_{21} & P_{31} & P_{41}\\P_{12} & P_{22} & P_{32} & P_{42}\\P_{13} & P_{23} & P_{33} & P_{43}\\P_{14} & P_{24} & P_{34} & P_{44}\end{bmatrix}_x$$

最后将式(9-56)展开得

$$\begin{aligned}x(s,t) &= (1-s)^3(P_{11_x}(1-t)^3+3P_{12_x}(1-t)^2t+3P_{13_x}(1-t)t^2+P_{14_x}t^3)\\&+ 3(1-s)^2s(P_{21_x}(1-t)^3+3P_{22_x}(1-t)^2t+3P_{23_x}(1-t)t^2+P_{24_x}t^3)\\&+ 3(1-s)s^2(P_{31_x}(1-t)^3+3P_{32_x}(1-t)^2t+3P_{33_x}(1-t)t^2+P_{34_x}t^3)\\&+ s^3(P_{41_x}(1-t)^3+3P_{42_x}(1-t)^2t+3P_{43_x}(1-t)t^2+P_{44_x}t^3)\end{aligned}$$

356

用同样方式可推导$y(s,t)$和$z(s,t)$。程序9-2要求的$X(s,t)$, $Y(s,t)$, $Z(s,t)$可直接由$x(s,t)$, $y(s,t)$, $z(s,t)$的表达式进行编码。画其他类型曲面的函数，也像Bézier曲面一样容易开发。

9.4 二次曲面

形如

$$f(x,y,z)=ax^2+by^2+cz^2+2dxy+2eyz+2fxz+2gx+2hy+2jz+k=0 \tag{9-57}$$

的隐式曲面方程定义了所有的二次曲面。比如，当$a=b=c=-k=1$且其他系数为零时，方程定义了一个以原点为中心的单位球体。当a到f都是零时，方程定义的是一个平面。在一些特定应用领域，比如分子造型[PORT79；MAX79]，二次曲面十分有用。同时，实体造型系统也集成了二次曲面。回忆一下，用三次有理曲线可以表示圆锥曲线；同样，用有理双三次曲面也可以表示二次曲面。因而，当只能用二次曲面表示时，可以用隐式二次方程表示有理曲面。使用二次曲面还有以下原因：

- 容易计算曲面的法线。
- 容易判断一个点是否在曲面上（只要将点坐标代入式(9-57)，然后检测其结果是否在零的某个ε领域内）。
- 给定x和y就很容易求出z（这在隐藏面算法中非常重要，见第13章）。
- 易于计算两个曲面的交。

式(9-57)也可以表示成

$$P^{\mathrm{T}}\cdot Q\cdot P=0 \tag{9-58}$$

其中

$$Q=\begin{bmatrix}a & d & f & g\\d & b & e & h\\f & e & c & j\\g & h & j & k\end{bmatrix} \qquad P=\begin{bmatrix}x\\y\\z\\1\end{bmatrix} \tag{9-59}$$

用Q表示的曲面做平移和放缩变换时也很方便。给定一个4×4的变换矩阵M（如第5章中介绍过的形式），变换后的二次曲面Q'由下式给出：

$$Q' = (M^{-1})^{\mathrm{T}} \cdot Q \cdot M^{-1} \tag{9-60}$$

由隐式方程 $f(x, y, z) = 0$所定义的曲面的法向量为[df/dx df/dy df/dz]。这要比9.3.4节讨论的双三次曲面的法向量更容易求解。 357

9.5 专用的造型技术

本章我们已经专注在几何模型方面，第10章还将继续。对于全由简单几何对象构成的世界而言，这些模型就足够了。但是对于许多自然景象用几何模型来表示并非是有效的，至少不能用于大的景象。例如，雾是由极小的水滴形成的，但如果采用每个水滴都是单独放置的模型，那么雾的造型是不可能的。再说，水滴模型并不能精确地表示我们对雾的感觉：在我们面前见到的雾，像是空气中的许许多多小点，而不是成千上万个水滴。雾在我们视觉中的感知是取决于它如何影响了到达我们眼睛的光线，而不是单一水滴的形状和放置。同样，树叶的形状可以由多边形、树干可以由管状样条来造型，但是通过显式地安放每一个树叉、树枝、细枝和树叶来造型一棵树是既耗时又笨拙的。

在[FOLE90]的第20章，我们可找到高级造型技术的深入讨论，这里我们将讨论两个专门的方法，它们非常容易实现，并且生成十分精彩的真实感图像。

9.5.1 分形模型

分形近来引起很多人的注意[VOSS87；MAND82；PEIT86]。分形图像是非常引人入胜的，而且已经开发许多产生分形图形的方法。术语分形在计算机图形学领域中被推广到包括在Mandelbrot原始定义之外的对象。现在，它的意思是任何对象只要它有准确或者统计自相似的测度，我们就可以称其为**分形**，当然精确的数学定义是要求在任何分辨率下都有统计的自相似性。因此，只有通过无穷递归生成的分形才是真正的分形物体。另一方面，通过有限步骤生成的物体，到某一阶段，在视觉上察觉不出细节上的差异，这是理想状态的适当近似。我们所说的**自相似性**可以利用一个例子即von Koch雪花很好地说明。开始时，一个直线段上有个凸包，如图9-27所示，我们对每一直线段都用与原始线段相似的线段替代。这个过程重复地进行：在图9-27b中每一段都用和完整形状相同的图替代。（用图9-27a的图形替代和用图9-27b的图形替代是一样的；如果使用图9-27a替代，经过2^n步的结果和在当前图形的每一段用每一阶段的整个图形替代的第n步的结果是相同的。）如果这个过程进行无限多次，结果就是所说的**自相似**：整个物体和它自身的一部分相似（也就是，可以被旋转、平移和缩放等）。

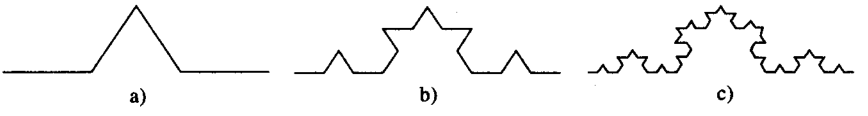


图9-27 von Koch雪花的构造：在a)中的每一段用整个图缩小3倍来替换，同样的过程应用于b)生成c)

和自相似性有关的概念是**分形维数**的概念。为了定义分形的维数，我们先看一下已知维数对象的一些特性。线段是一维；如果我们将线段分成N等份，每一部分和原始线段缩小一个比例因子$N = N^{1/1}$相似。正方形是二维的：如果我们把它分成N等份，每一部分和以因

子 $\sqrt{N}=N^{1/2}$ 缩小时的原始正方形相同。（例如，一个正方形被精确分为九个子正方形，每一个子正方形和原始正方形的1/3相似。）对于von Koch雪花又是什么情况呢？当它被分为四部分时（这些部分和在图9-27a中原始的四个线段有关），每一部分和原始图像的1/3相似。我们可以说它的维数是d，且$4^{1/d}=3$。d的值是log(4)/log(3)=1.26…。这就是分形维数的定义。

在这里值得提到的最著名的两个分形对象是：Julia-Fatou集和Mandelbrot集。这些对象产生于规则$x \to x^2+c$的研究（也包括许多其他规则，但这是最简单并且是最著名的）。这里x是**复数**[⊖]，$x=a+bi$。如果一个复数的模小于1，反复自乘将使它趋向于零。如果模大于1，反复自乘将使它越来越大。如果模等于1，经过反复自乘模仍为1。因此，一些复数反复自乘趋向于零，一些趋向于无穷大，还有一些既不趋于零也不趋于无穷大——这最后的一组数形成了趋于零的数和趋于无穷的数的边界。

假设我们对每一个复数x和不等于零的c反复使用映射$x \to x^2+c$，例如$c=-0.12375+0.056805i$；一些复数将会吸引到无穷大，一些将会吸引到有限的数，有些数将既不会吸引到无穷大，也不会吸引到有限的数。画出这些点的集合，我们就能得到如图9-28a所示的Julia-Fatou集。

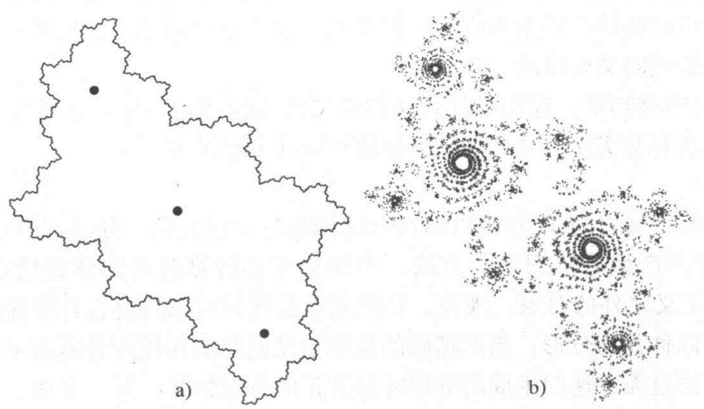


图9-28 Julia-Fatou集。a) $c=-0.12375+0.056805i$；b) $c=-0.012+0.74i$

应当注意图9-28b的区域不像图9-28a的那样保持了很好的连续性。在图9-28b中，一些点落向三个显示的黑点，一些趋向无穷大，还有一些与前两者相反。这最后的一类点是那些在图9-28b中被作为轮廓线画出的点。Julia-Fatou集的形状显然是由c的值决定的。如果我们计算所有可能c值的Julia-Fatou集，并且在Julia-Fatou集是连通的（也就是由一块组成，不是碎成不相连接的孤岛）时将点c着为黑色在该集不连通时c着为白色，我们就得到如图9-29所示的点集，称为**Mandelbrot集**。注意到Mandelbrot集是自相似的，即在集合的大圆盘的边界上，有几个小的集合，每一个和大的集合在比例上

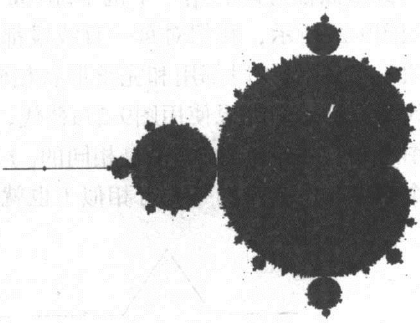


图9-29 Mandelbrot集。复平面上每一点c，如果对于过程$x \to x^2+c$的Julia集是连通的，就被着为黑色

⊖ 如果你不熟悉复数，把i当成一个特殊的符号并且知道复数的加法和乘法的定义就足够了。如果$z=c+di$是第二个复数，那么$x+z$定义为$(a+c)+(b+d)i$，xz定义为$(ac-bd)+(ad+bc)i$。我们可以把复数表示成平面上的点，将点(a,b)对应复数$(a+bi)$。复数$a+bi$的模是实数$(a^2+b^2)^{1/2}$，它给出复数大小的度量。

都是极为相似的。

幸运的是有一种非常容易逼近Mandelbrot集的方法：对每一个c值，取复数$0 = 0 + 0i$并且利用公式$x \to x^2 + c$有限次（比如1000次）。如果经过这么多次迭代值已经在定义的模小于100的圆盘之外，我们将c的颜色着为白色；否则为黑色。随着迭代次数和半径的增加，结果图像和Mandelbrot集就越接近。Peitgen和Richter [PEIT86]给出了Mandelbrot集和Julia-Fatou集的图像的生成方法的详细说明。

这些结果对于模拟自然形态是非常具有启发性的，因为很多自然现象都有惊人的自相似性。山有山峰、小山峰、岩石和碎石，它们所有的看起来都很相似；树有大枝、细枝和小枝，它们看起来也很相似；海岸线有海湾、水湾、河口、小河和灌渠，它们看起来也相似。因此，在某个尺度上建立自相似模型似乎是一种建立自然现象的壮观模型的方式。在此，自相似失效的尺度不是特别重要，因为意在建模而不是数学。因此，当一个物体经过足够多的步骤递归生成，以致该物体所有进一步变化都发生在像素分辨率以下，那么递归过程就没必要继续进行下去。

Fournier、Fussell和Carpenter[FOUR82]开发了一种基于递归分割生成一类分形山的方法。这在一维情况下最容易解释。假设我们以一个在x轴上的线段开始，如图9-30a所示。如果我们现在把线段平分为二等份并且在y方向把中点移动一个距离，我们得到图9-30b。继续分割每一个线段，我们从(x_i, y_i)到(x_{i+1}, y_{i+1})的线段中点计算一个新值：$x_{new} = (x_i + x_{i+1})/2$，$y_{new} = (y_i + y_{i+1})/2 + P(x_{i+1} - x_i)R(x_{new})$，这里$P()$是一个根据线段长度确定扰动幅度的函数，$R()$是一个根据$x_{new}$选择的在0至1之间的随机数[⊖]（见图9-30c）。如果$P(s) = s$，那么第一个点的位移不能超过1，接着的两个点的位移不能超过1/2（它们最大高度是1/2），依此类推。因此，所有的结果点都在单位正方形中。对于$P(s) = s^a$，结果的形状依赖于a的值；较小的a值将产生大的扰动，反之亦然。 358~361
当然，其他函数也可以使用，例如$P(s) = 2^{-s}$。

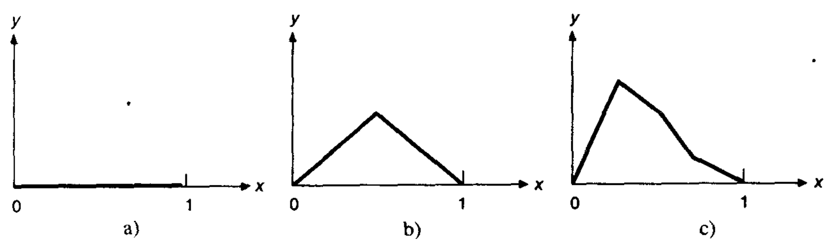


图9-30 a)在x轴上的直线段，b)直线的中点在y方向已经以一个随机量平移，c)进一步迭代的结果

Fournier、Fussell和Carpenter以如下方式使用这个过程来改变2D形状。他们从一个三角形开始，标记每条边的中点，并且连接三个中点，如图9-31a所示。每一个中点的y的坐标用我们以前描述的方式修改，导致如图9-31b所示的四个三角形。当迭代下去时，这个过程就会产生很真实的山，如彩图11所示（尽管从顶上看，会看到一个非常规则的多边形结构）。

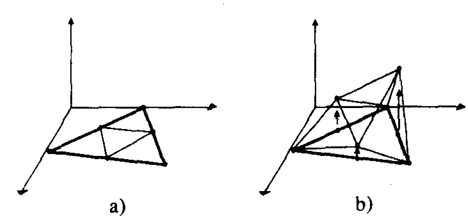


图9-31 a)一个三角形分割成四个较小的三角形，原始三角形的中点在y方向被扰动之后生成b)中的形状

⊖ $R()$实际上是一个随机变量，一个取实数值并产生0和1之间随机分布数的函数。如果由伪随机数发生器产生，就有一个优点，分形可以重复：我们可以通过提供同样的种子点给伪随机数发生器来在此产生它们。

注意到我们可以从具有一定形状的三角形的排列开始，然后应用这个过程生成进一步的细节。这种技术在某些建模应用时特别重要，比如在一个场景的物体布局中在较低层次看似是随机的，而在较高的层次上则是有规律的：在一个装饰花园中的植物可以采用随机机制产生，但
362 是它们的排列必须严格。另一方面，初始三角形排列的高层结构在迭代分割过程不变的话，对于某些应用是不合适的（特别是，如此生成的分形没有统计上的自相似，如在基于布朗运动的分形中显示的那样[MAND82]）。再有，由于任一个顶点的位置只被调整一次且以后就不变了，沿着原始三角形之间边沿的表面容易产生褶皱，这看起来很不自然。

绘制分形会是非常困难的。如果采用z缓存方式绘制一个分形，那么由于涉及大量多边形，显示起来将很费时。在扫描线绘制算法中，分类所有的多边形代价高昂，所以仅考虑与扫描线相交的多边形。但是由于必须判断每一个光线和数以百万计的多边形的相交情况，所以用光线跟踪来实现也是极困难的。Kajiya[KAJI83]对[FOUR82]中提出的分形给出了一种光线跟踪算法，Bouville[BOUV85]采用了更好的物体包围体而改进了该算法。

9.5.2 基于文法的模型

Smith[SMIT84]介绍了一个描述某些植物结构的建模方法，这种方法最初是由Lindenmayer [LIND68]开发的，它通过使用并行图文法语言（L文法）来描述，Smith称它为graftals。这种语言是通过一个产生式集合组成的语法来描述的，所有的产生式都至少用一次。Lindenmayer将语言扩展后包括了括号，因此语言的字母包括两个特殊的符号“[”和“]”。一个典型的例子是字母表{A，B，[，]}和以下两个产生式规则组成的文法：

1) A→AA

2) B→A[B]AA[B]

从公理A开始，前几代产生的是A，AA，AAAA，等等；从公理B开始，前几代是

0) B

1) A[B]AA[B]

2) AA[A[B]AA[B]]AAAA[A[B]AA[B]]

等等。如果我们说语言中的一个词代表图结构中一系列段，括号部分代表分支从哪开始，则由此生成的三层图如图9-32所示。

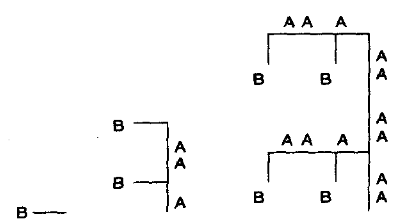


图9-32　语言的头三个词的树表示。所有的分支画在当前主轴的左侧

这个图的集合有一个很好的分支结构，但是更平衡一点的树会更有吸引力。如果我们将上述语言加入符号“(”和“)”，并改变第二个产生式为A[B]AA(B)，那么第二代为：

2) AA[A[B]AA(B)]AAAA(A[B]AA(B))

如果我们假设方括号代表左分支，圆括号代表右分支，那么相关的图如图9-33所示。在这样一种语言中，进行到后面，我们可以获得代表非常复杂模式的图结构。这些图结构有某种形式的自相似性，因为通过第n代的单词描述的模式（在此情况下，重复地）被包含到第$n+1$代中。

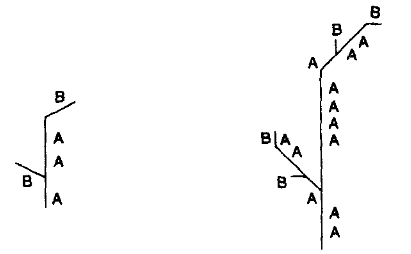


图9-33　两侧平衡的语言的头三个词的树表示。当前进到下一代时，我们已经使树的新的一段变短

从这样的词汇中生成一个物体是从生成词本身的过程中分离出来的。这里，树的各个部分以连续变短

的长度来绘制，每一个分支的角度都是45°，分支向左或向右生长。对于不同深度的分支选择不同的角度，对不同的段选择不同宽度的线(或者甚至不同直径的圆柱)给出不同结果；在树的每一个终节点绘制“花”或者“叶”会进一步增强图的效果。由于文法本身没有内在的几何内容，因此，使用基于文法的模型既需要语言的文法解释又需要语言的几何描述。

有好几个研究者对语言的增强和词汇的解释(也就是从词汇产生图)进行了研究[REFF88；PRUS88]。文法已经被丰富到允许我们跟踪在一个词汇中字母的年龄，使得老的和年轻的字母变换时是不同的（这种年龄的记录可由如下形式规则完成：A→B，B→C，C→D，…，Q→QG[Q]，以至于直到植物已经“老化”，感兴趣的过渡才发生）。许多工作都集中于构造文法，这些文法准确地表达植物发育的生物学。 363~364

然而，在某些情况下，一个文法作为植物的描述很笨拙：很多附加特征要加到文法或者文法词汇的解释中。在Reffye的模型[REFF88]中，植物生长的模拟是由一个小参数集来控制的，这些参数由生物术语来描述，并且体现在算法之中。该文法的产生式是按概率规则而不是确定性规则来使用的。

在下述模型中，我们像从前一样以一个简单的茎开始。在这个茎的顶端是**蓓蕾**，它可以经历几种变化：它可以死亡，可以开花然后死亡，可以休眠一段时间，或者成为一个**内部节点**——在两个蓓蕾之间的一段。一个内部节点的生成过程分为三步：一个原始的蓓蕾可以生成一个或者多个**叶腋蓓蕾**（在两个内部节点交叉处的某一侧的蓓蕾），这个过程称为**分支**；加入内部节点；新的内部节点的端点变为**顶点蓓蕾**（一个处在一系列内部节点端点处的蓓蕾）。图9-34a显示从蓓蕾到内部节点变化的例子。

在结果物体中的每一个蓓蕾都经历相似的变化。我们把树的原始部分称为具有**层次1**，我们可以采用演绎的办法定义其他内部节点的层次：由层次i的内部节点的顶点蓓蕾生成的内部节点也具有层次i；从层次i内部节点的叶腋蓓蕾生成的内部节点具有层次$(i+1)$。这样，整个树的主干是层次1，大枝为层次2，在大枝上的细枝为层次3，依此类推。图9-34b显示了一个更复杂的植物和这个植物不同内部节点的层次。

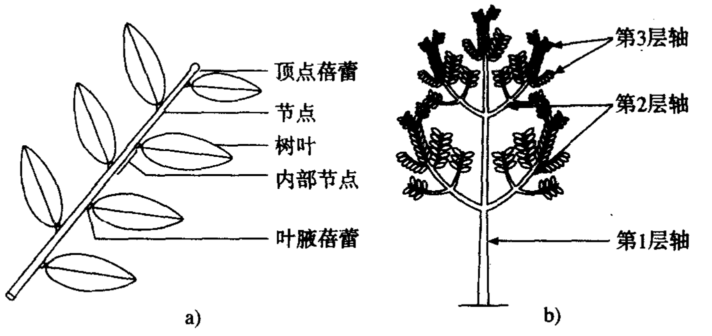


图9-34 植物生长的例子。a) 在一个植物的一段的顶点的蓓蕾可以成为内部的节点；之后，它产生一个新的蓓蕾（**叶腋蓓蕾**）、新的一段（**内部节点**）和一个在顶端的新蓓蕾（**顶点蓓蕾**）；b) 一个更复杂的植物，在不同的内部节点标有层次标记

将这种描述转换为一个实际树的图像需要一个模型，以描绘不同部分的形状：例如，层次1的内部节点可能是大的锥形圆柱，层次7的内部节点可能是细小的绿线。惟一的要求是在每一个叶腋节点上有叶子（尽管叶子有时会落下）。

最后，为了模拟这种模型中植物的生长，我们需要以下的生物学信息：模型的当前年龄，

每一个内部节点层次的生长速率，在每一个内部节点开始的叶腋蓓蕾数目（作为内部节点层次的函数），以及作为年龄、维数和层次函数的死亡、间歇、分支和迭代的概率。我们也需要一定的几何信息：每个内部节点的形状（作为一个层次和年龄的函数），每一个层次和年龄的分支角度，以及每个轴的取向（层次i的内部节点序列是一个直线还是朝向水平或垂直的曲线）。为了绘制一个植物的图像，我们还需要更多信息：每个要绘制的实体的颜色和纹理——不同层次的内部节点、不同年龄的树叶和不同年龄的花朵。极具有说服力的树模型可以通过基于文法的模型产生，见彩图12。

小结

本章涉及了关于曲线与曲面的表示的一些重要概念，运用这些表示实现交互系统，这些内容已经足够了。这些内容在理论方面的处理可参阅[BART87；DEBO78；FAUX79；MORT85]。

多边形网格是分段线性的，适合表示多面体，但对曲面体就不太适合。分段连续的三次参数曲线和双三次曲面广泛应用于计算机图形学和CAD中，用来表示曲面物体，主要是基于以下考虑：

- 单个x或y值允许有多个值与之对应。
- 可以表示无穷大斜率。
- 具有局部控制性，以便改变一个控制点仅仅影响到曲线周围一部分。
- 根据具体应用，可以插值或逼近控制点。
- 计算的有效性。
- 曲线或曲面的变换，可通过控制点的变换获得。

虽然这里只讨论了三次曲面，还可以使用高于三次或低于三次的曲面。前面提到的内容对一般次数n的参数曲线和曲面都适用。

365~366 我们也简要地讨论了一些自然现象建模的技术，特别是分形和基于文法的方法。

习题

9.1 求出式(9-11)中定义的参数表示的直线的几何矩阵和基矩阵。

9.2 证明，若平面曲线$[x(t)\quad y(t)]^T$具有G^1连续性，则曲线段拼接点两侧的斜率dy/dx相等。

9.3 令$\gamma(t)=(t, t^2)$，其中$0\leqslant t\leqslant 1$，并令$\eta(t)=(2t+1, t^3+4t+1)$，其中$0\leqslant t\leqslant 1$，注意$\gamma(1)=(1,1)=\eta(0)$，所以γ和η在拼接点处C^0连续。

a. 画出曲线$\eta(t)$和$\gamma(t)$，$0\leqslant t\leqslant 1$。

b. $\eta(t)$和$\gamma(t)$在拼接点处具有C^1连续性吗？（计算向量$d\gamma/dt(1)$和$d\eta/dt(0)$来验证你的答案。）

c. $\eta(t)$和$\gamma(t)$在拼接点处具有G^1连续性吗？（计算b中两个向量的比值来验证你的答案。）

9.4 曲线$\gamma(t)=(t^2-2t+1, t^3-2t^2+t)$和$\eta(t)=(t^2+1, t^3)$，定义在区间$0\leqslant t\leqslant 1$内。由$\gamma(1)=(1,0)=\eta(0)$知两曲线相接。证明两曲线在拼接点处具有$C^1$连续性，但没有$G^1$连续性。以$t$为变量画出两条曲线以验证为什么这个行为发生。

9.5 证明两条曲线$\gamma(t)=(t^2-2t, t)$和$\eta(t)=(t^2+1, t+1)$在$\gamma(1)=\eta(0)$处都是C^1和G^1连续的。

9.6 分析四个相邻的控制点共线对B样条曲线的影响。

9.7 写一个程序，可以输入任意一个几何矩阵、基矩阵和控制点列表，并画出相应的曲线。

9.8 求两条相接的Hermite曲线具有C^1连续性的条件。

9.9 假设关于Hermite几何条件和Bézier几何条件的等式为$R_1=\beta(P_2-P_1)$，$R_4=\beta(P_4-P_3)$。给定四个均匀分布的Bézier控制点$P_1=(0,0)$，$P_2=(1,0)$，$P_3=(2,0)$，$P_4=(3,0)$。证明：若要使参数曲线$Q(t)$从P_1到P_4具有恒定的速率，则系数β必须等于3。

9.10 说明为什么均匀B样条曲线的式(9-35)要写成$Q_i(t-t_i)$，而非均匀B样条曲线的式(9-37)写成了$Q_i(t)$?

9.11 已知平面非均匀B样条曲线以及曲线上的一点(x, y)，写一个程序求出相应的t值。必须考虑到一种情况，对一个给定的x值（或y值），可能有多个对应的y值（或x值）。

9.12 用推导Hermite曲面表示式(9-49)和式(9-50)的方法推导出Bézier曲面的表示式(9-51)。

9.13 令$t_0=0$, $t_1=1$, $t_2=3$, $t_3=4$, $t_4=5$。用这些值求出$B_{0,4}$和其定义中的每一个权函数。然后再在区间$-3\leqslant t\leqslant 8$上画出这些函数。

9.14 类似于例9.2，采用程序9-2的框架，开发一个显示Bézier曲面片的程序。该程序应该提供显示指定曲面片控制点的选项，这样用户可以（通过使用定位器）选中任一控制点并移它到新位置。然后应该重画曲面片以反映新的几何约束。由于定位器是二维输入，你如何将它和三维控制点联系起来？你也可以指定自己的曲面片几何数据或现成的数据，像图9-1的茶壶那样。[CROW87]中有茶壶的完整数据。 367 368

第10章 实体造型

第9章所讨论的表示方法可以描述二维和三维的曲线和曲面。正像2D直线和曲线的集合不足以描述一个闭区域的边界一样，一个3D平面和曲面的集合也不能够围成一个封闭的体。但在许多应用中，重要的是能够区分一个3D形体的内部、外部及其表面，这样才能导出形体的许多性质。例如，在CAD/CAM领域，一个实体如果能充分利用它的几何特征来建模，那么在制造它前，就可以对它施行许多操作。我们可能希望能判定两个形体彼此是否相交，例如，机器人的手臂是否会碰到其他物体，或者，切削刀具是否只切削预定的材料。在机械仿真中，重要的是计算物体（如齿轮）的特性，如体积和质心等。在实体造型中应用有限元分析，就是通过有限元造型来确定诸如应力和温度等因素。实体的一种合适的表示甚至可能在实体的数控加工制造中为刀具自动生成加工指令代码。另外，像产生折射透明等一些图形学技术，与光线从何处进入和穿出实体的判断有关。这些应用都是**实体造型**的例子。构造实体的需求促进了表示实体的各种方法的发展。本章将简述这些表示方法。 369

10.1 实体表示

把形体表示成看起来像实体，这并不表明，这种表示能力对实体的表示是合适的。考察一下到目前为止，我们是怎样表示形体的，把实体看成是直线段、曲线、多边形和曲面等的罗列。如图10-1a所示，其中的线段定义了一个立方体了吗？如果把形体每一侧的四条线段看成是构成四边形面的边界，那么这个图所表示的就是立方体。但是在这种表示方式中没有任何理由要求对直线段做这样的解释。例如，把任一这样的面去掉，同样的这些直线段可用于画这个图。在画图中，如果我们把连接直线段的每一个平面环定义成一个多边形面，又会怎样？在图10-1b中，由图10-1a中所有的面，再加上一个额外的悬挂面，会产生一个不能界定其体积的形体。在10.5节我们将看到，如果想保证所表示的造型就是实体，那么对实体的定义需要一些额外的约束。

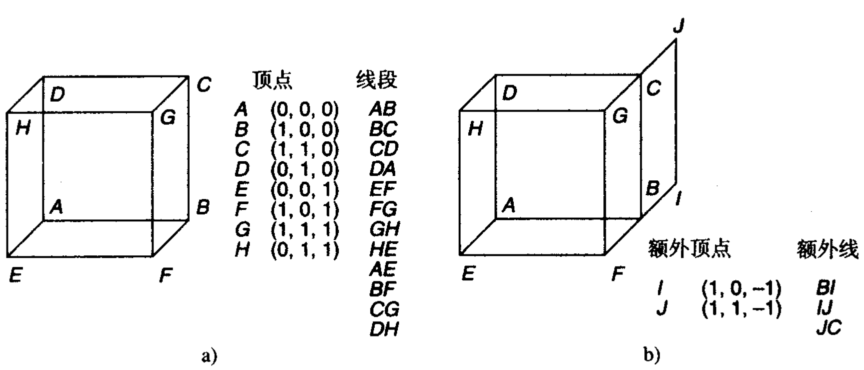


图10-1 a) 由12条直线组成的线框立方体，b)有一个额外面的线框立方体

Requicha[REQU80]给出了实体的表示形式应具有的性质。表示的**域**应足够大，以允许表示所需的实际形体。理想的表示方式应该是*无歧义的*，即不像图10-1a所表示的形体那样，会有“想要表示什么样的形体”这样的疑问，并且给定的表示应该对应一个且仅一个实体。无歧

义的表示也称为是**完备的**。如果一种表示可以仅用一种方式把给定的实体编码表达出来，那么这样表示是惟一的。如果一种表示可以保证惟一性，则像“判别两个形体是否相同”这样的操作就容易了。精确表示是不用逼近方式来表示形体。正像许多只能画直线段的图形系统用逼近
370 方法表示光滑曲线一样，一些实体造型表示方式也是用逼近方式表示形体。一种理想的表示方法不应产生无效的表示（所谓无效表示是指它没有与相应的实体相对应），如图10-1b所示。另一方面，借助于交互实体造型系统，它应该容易生成有效表示。我们希望一个形体在旋转、平移和其他操作下保持**封闭性**，即在有效形体上进行这些操作，其结果应仍然是有效的。表示方式还应该是紧凑的，便于节省存储空间，同时在分布式系统中可以减少通信时间。最后，表示方式应允许使用高效的算法以计算指定的物理性质，对我们而言最重要的是生成图像。

事实上，很难找到这样的表示方法，能满足上述所有这些性质，通常要有所折衷。由于这里只讨论目前正在使用的几种主要的表示方式，重点是提供足够的细节，以便能够理解这些表示方式是如何在图形软件中体现出来的。实体造型方面的更多的细节可参见[REQU80；MORT85；MÄNT88]。

10.2 正则布尔集合运算

不管形体是如何表示的，我们希望能把它们拼合起来，以生成新的形体。最直观和最常用的方法是利用集合论中的**布尔集合运算**并、差、交等，如图10-2所示。三维情形与二维情形类似。但是，对两个实体进行通常的集合运算，不一定能得到一个实体。例如，只有一个顶点相交的两个立方体进行通常的集合运算得到的是一个点。

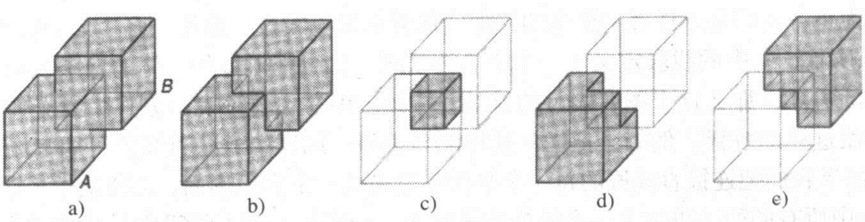


图10-2 布尔运算：a)实体A和B，b)$A\cup B$，c)$A\cap B$，d)$A-B$，e)$B-A$

我们用**正则布尔集合运算**[REQU77]来代替通常的布尔运算，用$\cup^*$、$\cap^*$和$-^*$来表示，这样，两个实体进行布尔运算后总能生成实体。例如，只有一个顶点相交的两个立方体的正则求交运
371 算结果是空集。

为了探讨两种布尔运算之间的差别，我们把形体看成是一个由内部点和边界点构成的点集，如图10-3a。**边界点**集由一个形体到它的补集之间的距离为0的点集构成。边界点不一定是形体上的点。一个形体的**闭集**包含它的所有边界点，而**开集**则不包含。一个点集与它的边界点集的并集称为这一集合的**闭包**，如图10-3b所示。闭包本身也是闭集。一个闭集的**边界**是其边界点的集合，而它的**内部**则是由这个集合除边界以外的所有其他点构成，如图10-3c所示，因此，相对于这一形体而言，内部是边界的补集。集合的**正则化**定义为这个集合的内部点的闭包。图10-3d所示就是图10-3c中形体的闭包，因此，就是图10-3a所表示的形体的正则化。若一个集合与它的正则化相同，则这个集合被称为**正则集**。注意正则集不包含那些与点集内部点不相连的边界点；因此它没有悬挂的边界点、线或者面，与普通的集合运算相对应，正则的集合运算符可以定义如下：

$$A\ op^*\ B = \text{closure}(\text{interior}(A\ op\ B)) \tag{10-1}$$

式中op表示∪、∩或者－。对正则集应用正则集合运算符时，只生成正则集。

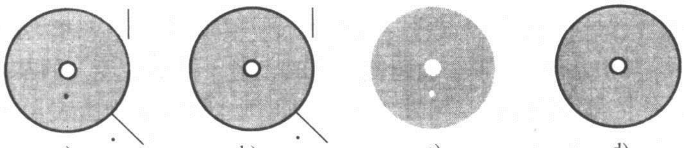


图10-3 形体的正则化。a)形体由内部点和边界点组成。内部点由浅灰色表示，边界点的一部分由黑色表示，其余部分由深灰色表示。该形体含有悬挂的和分离的点、线，并且在形体的内部还有一个边界点，这一点不是形体中的点。b)形体的闭包。形体的所有边界点都属于该形体。a)中在形体内部的一个边界点已属于形体的内部点。c)形体的内部。悬挂的或分离的点和线段都已删除。d)形体的正则化集是形体内部的闭包

现在我们来比较对正则集进行普通集合运算和正则集合运算时，两者之间的差别。考虑两个形体（如图10-4a所示），两形体放置如图10-4b所示。两形体的普通布尔交集包括每个形体的内部和边界与另一个形体的内部和边界的交集，如图10-4c所示。相反，两形体的正则布尔交集（如图10-4d所示）则包括两者的内部的交集以及一个形体的内部与另一个形体的边界之间的交集，后者仅仅是两形体的边界交集的一个子集。用于确定这个子集的准则决定了为何正则布尔运算与普通布尔运算的求交运算之间有差别，后者包括了两集合边界的交集的所有部分。

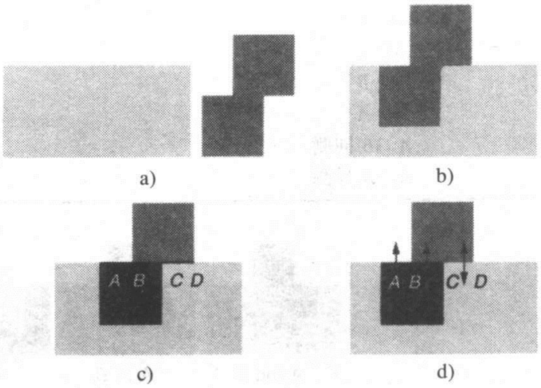


图10-4 布尔交集。a)两个形体的截面；b)两形体求交前的位置；c)普通布尔求交运算导致了一个悬挂面，如图截面中的线段CD；d)如果两形体位于它的同一侧，正则布尔交集包含了一部分共享边界(AB)；如果形体位于边界的异侧，不包含一部分共享边界(CD)。边界与内部的交集(BC)总是属于正则交集内的

直观上看，当且仅当两形体的内部位于这段共享边界的同一侧，两形体边界与边界的一段交集才属于正则集合运算的交。因为两形体与那段共享边界直接相连的那些内部点集是两形体的交集，这一段边界也肯定属于这些内部点集的闭包内。考察两多面体的一侧表面共面时的共享边界情形。如果定义了两形体表面的向内（或向外）的法向，那么，判断两形体的内部点集是否位于它们的共享边的同一侧就简单了。若它们的法向是同向的，则内部点集在同侧。因此图10-4d中的AB段就属于正则交集。记住，如果一个形体的边界与另一个形体的内部相交，如图10-4d中的BC线段，则这部分边界总是属于正则交集的。

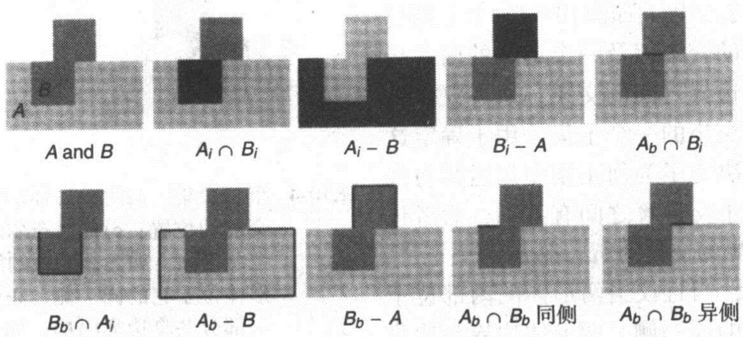
再来考察一下，当两形体的内部点集分别位于它们的共享边的异侧时，会怎样？如图10-4d中CD段情形，与这条边相邻的内部点集都不属于两形体的交集。因此，这段共享边与两形体的交集的内部不相连，从而不属于两形体的正则交集。对共享边属于交集进行这一额外的限制，保证了交集是正则集。交集边界上的每一侧面的法向是原形体边界中的成为交集边界部分的面法向。（在第14章将会看到，在形体的消隐中，面的法向很重要。）确定了哪些面在边界上后，如果边界与边界的交集中的边或顶点与这些面相连，那么，这些边或顶点也属于交集的边界。 372~373

每一种正则运算可以通过对形体的边界和内部进行普通集合运算来定义。表10-1表示如何

对形体A、B作正则集合运算。图10-5表示运算结果。A_b和A_i分别表示A的边界和A的内部。$A_b \cap B_b$ 同侧表示A和B的共享边界的一部分，其中A_i和B_i在它的同一侧。也就是说，对于共享边界上的某一点b，如果至少有一点内部点i与之相近，则点i同时属于A和B。$A_b \cap B_b$异侧表示A和B共享边界的一部分，其中A_i和B_i位于它的两侧。此时，对于共享边界上的某一点b，找不到这样的内部点i与之相近，使得i同时属于A_i和B_i。每一种正则运算定义为由表10-1中所在列中带·号项所对应的普通集合运算的并集。

表10-1 正则布尔运算

集合	$A \cup^* B$	$A \cap^* B$	$A -^* B$
$A_i \cap B_i$	·	·	
$A_i - B$	·		·
$B_i - A$	·		
$A_b \cap B_i$		·	
$B_b \cap A_i$		·	·
$A_b - B$	·		·
$B_b - A$	·		
$A_b \cap B_b$同侧	·	·	
$A_b \cap B_b$异侧			·



374 图10-5 两形体的子集之间的普通布尔运算

注意，对于每一种情形，正则运算所生成的边界中的每一部分，或者在原一个形体的边界上，或者在原来两个形体的边界上。在计算$A \cup^* B$或者$A \cap^* B$时，计算得到的形体的表面的法向是原来一个形体或者两个形体所相应的表面的法向。但在$A -^* B$情形中，运算的结果通常是A被B挖去一部分，这一部分的每一表面的法向一定是与B在这一表面上的法向方向相反。这一点与$A_b \cap B_b$ 异侧和$B_b \cap A_i$ 的边界部分得到的结果相同。或者，$A -^* B$也可写成$A \cap^* \bar{B}$。$\bar{B}$（B的补集）可以通过对B的内部取补以及对B的边界处的法向取反得到。

作为一种用户界面技术，正则集合运算已用于大多数从简单形体生成复杂形体的造型表示方法中，这些造型方法将在后面讨论。构造实体几何这一造型表示方式显然也包含这一操作手段。在以下几节中，我们将详细叙述几种无歧义地表示实体的方法。

10.3 基本实体举例法

在**基本实体举例**（primitive instancing）中，造型系统定义一个基本的三维实体形状的集合，它们与应用领域有关。这些基本实体用典型的参数形式表示，而不用第7章所讨论的变换，

但也有其他性质。例如，一个基本实体可能是一个正规的棱锥体，具有与顶点相连的面数，面数由用户指定。基本实体举例类似于参数化对象，如第2章的菜单，只是这里的对象是实体。参数化基本实体可看成定义一个零件类，这个类的数目随各种参数而变化，这是一个重要的CAD概念，称为**群组技术**。基本实体举例法常用于一些相对复杂的对象，如齿轮或者螺栓等。如果它们用更简单的实体通过布尔并运算来生成，那就太繁琐了，所以用一些简单的参数来刻画。例如，齿轮可以用它的半径或者齿数来参数化表示，如图10-6所示。

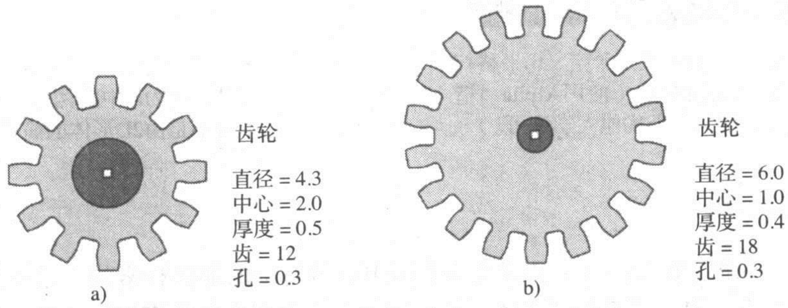


图10-6 由基本实体举例法定义的两个齿轮 375

虽然可以建立基本实体举例的层次结构，但每一叶子节点仍然是一个单独定义的对象。在基本实体举例法中，不能通过基本实体的组合（例如利用正则的集合运算）来生成更高层次的基本实体。因此，生成新的基本实体的惟一方法是通过编写代码来定义。类似地，绘制实体图或者是确定其质量等性质的例程必须对每一个基本实体单独编写。

10.4 扫掠表示法

一个形体沿空间路径做扫掠运动，其运动轨迹定义了一个新的形体，这一过程称为**扫掠**（sweep）。最简单的扫掠是由一个2D区域沿区域所在平面垂直的线性路径进行扫掠生成一个体来定义的。这种体被称为**平移扫掠**体或者**拉伸**。它可以自然表示一类形体，可以通过给定的截面形状沿模板拉伸金属或塑料制品而生成的形体。在这些简单的情形中，每个扫掠体的体积就是扫掠体的截面积与扫掠长度的简单乘积。简单的推广涉及截面在拉伸过程中可以缩放，生成带斜度的形体，或者截面沿着一条与截面所在平面不垂直的线性路径扫掠。**旋转扫掠**是一个区域绕轴旋转生成的形体。图10-7表示两个形体以及由此生成的简单的平移和回转扫掠体。

被扫掠的形体不一定是二维的。实体的扫掠在机床刀具移动或者机器人沿路径运动所生成的区域构造中很有用。扫掠体在扫掠过程中，它的面积或者体积在大小、形状或者扫掠方向等方面都可改变，扫掠路径也可以是任意的曲线，这种扫掠称为**一般扫**。2D区域的一般扫在计算机视觉中[BINF71]称为**广义柱面**，它的构造是参数化的2D截面沿任意曲线运动生成，且截面线始终与路径垂直。高效的一般扫生成方法是十分困难的。例如，路径和形体形状选择可能使得扫掠体自交，从而使体积计算很复杂。同样，一般扫也不总能生成三维实体。例如，一个2D区域沿它所在平面做扫掠，生成的是另一个2D区域。

一般地，如果事先不转换成其他某一种表示，对扫掠体施加正则集合运算是很困难的。在正则集合运算中，即使是最简单的扫掠体也是不封闭的。例如，两个简单的扫掠体的并，一般不是一个简单的扫掠体，如图10-8所示。虽然扫掠存在着封闭性和计算等问题，但它仍然是一种自然的和直观的构造各种实体的方法。因此，许多实体造型系统允许用户构造扫掠体，但存储时却是我们要讨论的另外一种表示形式。

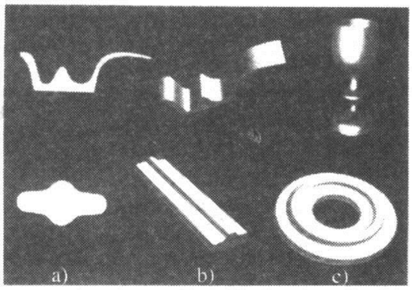


图10-7 扫掠。a) 2D区域用于定义b)平移扫掠和c)旋转扫掠。（使用Alpha_1造型系统创建，经犹他大学许可。）

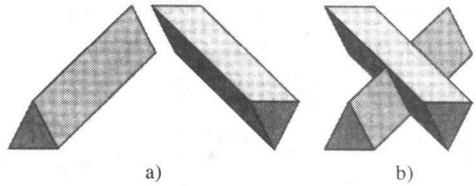


图10-8 a)两个由2D形体（三角形）构成的简单扫掠体，b)为a)中的两个扫掠体的并集，它本身不是由2D形体的简单扫掠体

10.5 边界表示法

边界表示（也称为b-reps）有点儿像我们在10.1节讨论的那种朴素的表示方法，它是用实体表面边界的顶点、边、面来刻画实体。某些边界表示限制在平面多边形边界，甚至要求面是凸多边形或者是三角形。如果允许用曲面来表示，那么要确定
376~377 表面的构成就很困难了，如图10-9所示。曲面常用多边形来逼近。实体也可以用曲面片来表示，如果处理这些表示的算法能处理交线，而这样的交线一般会比原曲面的次数要高。边界表示由前面几章介绍的简单的向量表示发展而来，并且已用于目前许多造型系统中。由于它在图形学中被广泛采用，现已开发了大量的高效的技术来生成多面体实体的光滑消隐图，其中一些技术将在第14章讨论。

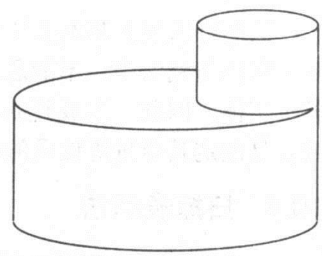


图10-9 这一实体共有多少个面

许多边界表示系统只支持边界是用**二维流形**表示的实体。根据定义，二维流形上的每一点存在一个任意小的邻域，使得这个小邻域与欧氏平面上的一个圆盘是拓扑等价的。也就是存在一个从小邻域与圆盘之间的连续的一一对应关系，如图10-10a和图10-10b所示。例如，如果有多于两个面共享一条边，如图10-10c所示，则这条边上一个点的任一邻域包含这些面中任一个面上的点。显然没有从这个邻域到平面圆盘之间的一一对应。但这一点从数学上证明并那么直接。因此图10-10c中的表面不是二维流形。虽然现在有些系统并没有这样的限制，但我们仍然把讨论范围限定在边界是二维流形上，除非在有的地方特别声明。

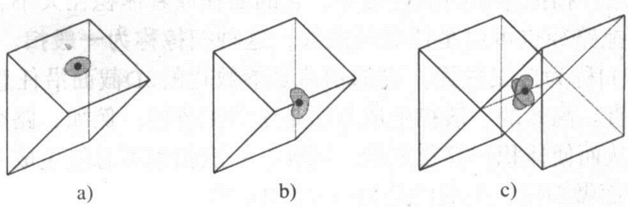


图10-10 在二维流形上，每个点（显示为黑点）都有一个周围的点的邻域（它是一个拓扑圆盘），如a)和b)中灰色部分所示。c)如果物体不是一个二维流形，那么它具有没有邻域（它是一个拓扑圆盘）的点

10.5.1 多面体和欧拉公式

多面体是由一组多边形作为边界构成的实体，它的每一条边属于偶数个多边形（在二维流形

情形中只有两个多边形），它也要满足其他的一些约束（这一点将在后面讨论）。**简单多面体**是这样一个实体：它可以通过形变变成一个球形，也就是说，与环状体不同，这样的多面体没有洞。简单多面体的边界表示满足欧拉公式，它表示简单多面体的顶点数、边数和面数之间的一种不变关系：

$$V - E + F = 2 \tag{10-2}$$

式中V是顶点数，E是边数，F是面数。图10-11表示一些简单多面体以及它们的顶点数、边数 378
和面数。注意，如果有曲线边和非平面面，欧拉公式仍成立。欧拉公式只说明了一个实体是简单多面体的一个必要条件而非充分条件。可以构造这样的形体，它满足欧拉公式，但没有围成一个体，这可以在一个有效的实体上附加一个或多个悬挂面或悬挂边来构造，如图10-1b所示。因此，有必要加上约束以保证形体是实体：每一条边必须与两个顶点相连，每一条边只有两个面共享，每一个顶点至少有三条边，面之间不能相互贯穿。

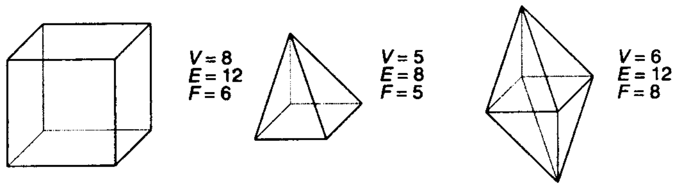


图10-11 多面体以及它们V，E和F的值，每一种情形，满足$V - E + F = 2$

对于面上带洞的二维流形，有广义欧拉公式：

$$V - E + F - H = 2(C - G) \tag{10-3}$$

式中H是面上洞的数目，G是贯穿形体的洞的数目，C是形体相离部件的数目，如图10-12所示。如果实物是单部件的，那么其中的G称为**亏格**；如果是多部件的，那么G是每一部件上亏格的和。与前面一样，附加的约束条件也必须保证形体是实体。

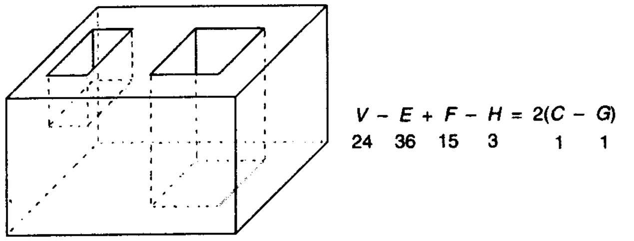


图10-12 多面体按式(10-3)分类，它的顶面有两个洞，底面有一个洞

Baumgart引入了**欧拉算子**的概念，它对满足欧拉公式的实体做运算，将实体转换成新的实体， 379
也满足欧拉公式，方法是增加和删除顶点、边和面[BAUM74]。Braid、Hillyard和Stroud[BRAI78]说明了如何用少量的欧拉操作组合来生成新的实体，并指出，中间结果可以不必是有效实体。同时Mäntylä[MÄNT88]证明了所有有效的边界表示都可以通过有限步的欧拉操作来构成。另外也可以定义一些不影响实体的顶点、边、面数目的操作，它们通过移动现有的顶点、边和面来提拉形体。

最简单的边界表示可能是列出所有的多边形面，每一个面由顶点坐标列表表示。为了表示每个多边形面的方向，多边形顶点从实体外部看按顺时针方向排列。为避免被面共享的点的重复表示，在坐标列表中引入索引来表示面的顶点。在多边形顶点列表中，这种表示把边隐式地表示成两相邻的顶点。对于每个面，边不是显式地表示成顶点对，而是在边表中定义成索引列表。这些表示方法已在9.1.1节中详细讨论了。

10.5.2 布尔集合运算

利用正则的布尔集合运算，边界表示可以组合生成新的边界表示[REQU85]。Sarraga[SARR83]和Miller[MILL87]讨论了确定二次曲面间的求交算法。[TURN84；REQU85；PUTN86；LAID86]中给出了多面体组合的算法，Thibault和Naylor[THIB87]描述了一种基于空间划分二叉树的实体表示方法，这种表示方法将在10.6.4节中讨论。

[LAID86]中的一种方法是检查两形体的多边形，如果需要，则可对它们进行分割，以保证确定两形体之间的点、边、面的交是它们的点、边或者是面。然后，一个形体的多边形将依据另一形体分类，以确定它们是在另一形体的内部、外部或边界上。回顾一下表10-1，注意到，由于它是边界表示，我们只要考虑表中最后的六行，其中每一行表示原来两形体边界A_b和B_b之一或者两者中的一部分。分割后，每一形体的每个多边形或者全部在另一形体的内部（$A_b \cap B_i$或$B_b \cap A_i$），或者全部在另一形体的外部（$A_b - B$或$B_b - A$），或者是共享边界的一部分（$A_b \cap B_b$同侧或$A_b \cap B_b$异侧）。

多边形也可以根据光线投射技术分类，这一技术将在13.4.1节讨论。从多边形的内部一点沿多边形的表面法向构造一个向量，然后找出另一形体中与这一向量有交，且相距最近的多边形。如果没有与此向量相交的多边形，则原多边形在另一形体的外面。如果最近的相交多边形与原多边形共面，此时就是边界与边界的交，比较两者的法向可以知道是哪一类的交（$A_b \cap B_b$同侧或者$A_b \cap B_b$异侧）。否则要检查两个多边形法向的点积。点积为正，说明原多边形在另一形体的内部；点积为负，则说明在另一形体的外部。如果向量在相交多边形的平面上，则点积为0；对于这一情形，向量被轻微的扰动，再与另一形体求交。
380

顶点相邻的信息，可以用来避免用这种方式对每一多边形分类这样的开销。如果一个多边形与一个分过类的多边形相邻（即与之同享顶点），并且与另一形体的表面没有交，则它可以归入同一类。在多边形分割的初始阶段，两形体的共同边界上的所有顶点可以做上标记。一个多边形与另一形体的表示是否有交可以通过检查它是否有边界顶点来确定。

每一多边形的分类决定了它在生成复合形体的操作过程中是否被保留，如10.2节所述。例如，在构成集合的并时，一个形体的任一多边形如果在另一形体的内部，则不予考虑。两形体之一的任一多边形如果不在另一形体内部，则这一多边形予以保留，除非是多边形间的共面情形。如果两个共面多边形的表面法向是反向的，则两者不予考虑；如果表面法向是相同的，则只保留两个多边形中的一个。如果形体是由不同材料构成的，则确定哪个多边形予以保留这样的判断就很重要。虽然$A \cup^* B$与$B \cup^* A$有相同的几何意义，但两者看起来可能会有差别，所以在多边形共面情形，两者可根据自己的喜好，选择其中之一作为运算的定义。

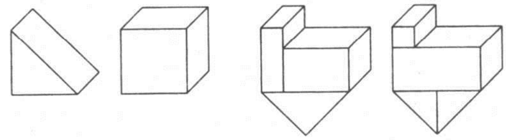
10.6 空间划分表示法

在**空间划分**表示中，实体被分解成相连的但不相交的实体组合，它比原来的实体有更多的基本实体，尽管不必与原来实体类型相同。基本实体可以在类型、大小、位置、参数化或方向等方面各不相同，很像小孩子搭积木时用的不同形状的积木。对形体分解到何种程度依赖于实体需要怎样的基本实体以便于准备完成感兴趣的运算。

10.6.1 单元分解法

空间划分中最一般的类型之一是**单元分解**。每一个单元分解系统定义了一个基本实体单元集，这些基本实体单元已经参数化，并且常常含有曲线边。单元分解法与基本实体举例法不同，这里，我们可以用简单的、基本的单元自底向上把它们“胶合”在一起复合成更为复杂的形体，

胶合操作可理解为严格的并，其中的形体不能有交。对胶合单元的更进一步的限制是，两个单元共享一个点、一条边或一个面。虽然一个形体的单元分解表示是明确的，但不一定是惟一的，如图10-13所示。单元分解的有效性证明也较困难，因为每一对单元都必须进行可能的求交测试。不过，在有限元分析中，单元分解法是一种重要的表示方法。



a) b) c)

图10-13 用不同方法变换a)显示的单元，可构造b)及c)显示的同样形体。即使一种单元类型也足以引起歧义性

10.6.2 空间位置枚举法

空间位置枚举（spatial-occupancy enumeration）是单元分解的一种特殊情形，这里实体被分解成相同的单元，并以固定的、正则的网格排列。这些单元常称为*体素*，类似于像素。图10-14表示了一个被空间位置枚举法表示的形体。最常见的单元类型是立方体，以正规的立方体排列的空间表示方式，被称为cuberille。用空间位置枚举法来表示形体时，只要控制网格上每一位置单元是否为空。在表示一个形体时，我们只须判断形体包含哪些单元，不包含哪些单元即可。因此，形体可以用惟一的且明确的所占单元列表方式表示出来。判断一个单元在实体的内部或外部是很容易的，要判断两个实体是否邻接也同样简单。空间位置枚举法常应用于生物医学领域，用来表示体数据，这些数据来自计算机X射线轴向分层造影（CAT）扫描等途径。

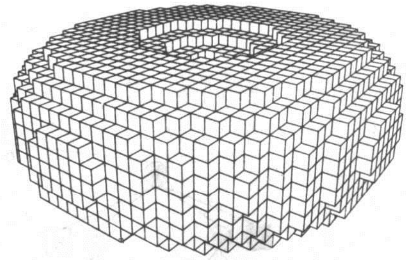
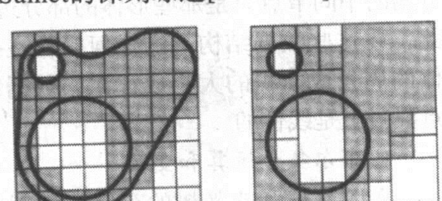


图10-14 由空间位置枚举法表示的圆环（摘自SIGGRAPH'80 Conference Proceedings, *Computer Graphics* (14)3, July 1980，作者A.H.J.Christensen，由ACM公司许可使用。） 381~382

尽管空间位置枚举方法有很多优点，但它也有许多明显的缺陷，这些缺陷类似于二维形状表示法，后者只用一位深度的位图表示。它没有"部分"充满的概念。因此，许多实体只能近似表示，图10-14就是一例。如果单元是立方体，则只有那些表面与单元立方体侧面平行并且顶点严格落在网格点上的实体，才能被精确表示出来。类似于位图中的像素，原则上，单元可以被分割成预定的大小，以增加表示精度。但是空间开销成了一个重要问题，因为在三维空间中，当体素分辨率为n时，表示一个形体就需要占用高达n^3个单元。

10.6.3 八叉树表示法

八叉树在表示层次上改进了空间位置枚举，以满足存储要求。八叉树实际上由**四叉树**而来，四叉树是一种二维表示格式，用于图像编码。正像Samet的深刻综述[SAME84]所说，这两种方法都是由许多研究人员独立发现的，四叉树在20世纪60年代后期至70年代早期出现[WARN69；KLIN71]，八叉树是在20世纪70年代后期至20世纪80年代早期出现[HUNT78；REDD78；JACK80；MEAG80；MEAG82]。



a) b)

图10-15 形体表示，a)用空间位置枚举法表示，b)用四叉树表示

四叉树和八叉树的基本思想都是二分法中的分而治之的方法。四叉树是对二维平面在两个方向上进行连续分割以形成象限的方式得到，如图10-15所示。当

用四叉树来表示一平面区域时，根据象限与细分的平面区域相交的程度，每一个象限是充满平面区域、部分充满或者为空（也分别称为黑色、灰色和白色），部分充满的象限再递归地细分成子象限，直到所有的象限类型相同（满或者空）或者细分到预先设定的中止深度。当四个同一层的相邻象限都是充满或者都是空时，删除这些象限并且让它们上一层从部分充满的父象限改成充满或者为空（用自底向上的方法，可以避免删除和合并操作[SAME90b]）。在图10-15中，在中止深度上所有部分充满的节点归入充满的节点类中。递归细分的过程可以用一棵树来表示，其中部分充满的象限作为中间节点，而充满和空的象限都为叶节点，如图10-16所示。这一思想可以与13.5.2节中讨论的Warnock区域细分算法进行比较。如果把节点归类的标准放宽，把在标准上下的节点都归类到或者是充满的或者是空的两类中，则区域表示将更加简洁，但表示精度却降低了。八叉树表示方式与四叉树类似，只是递归细分时是沿三个方向分割象限，如图10-17所示。

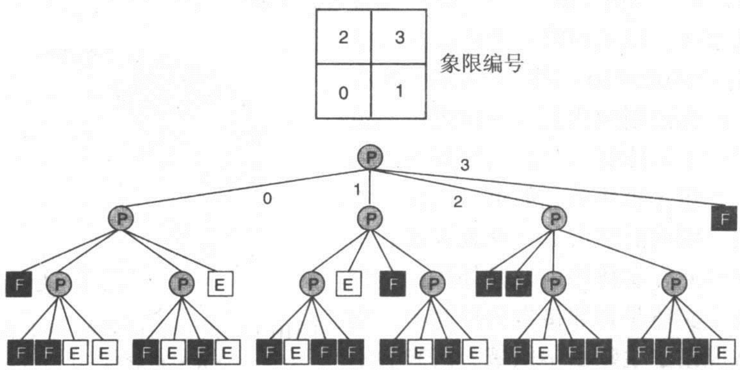


图10-16 图10-15中所示形体的四叉树数据结构。F = 充满，P = 部分充满，E = 空

四叉树的各个象限常用数字0到3来表示，而八叉树则用数字0到7来表示。由于没有标准的编号方式，也可以用一些助记符号。四叉树的象限是根据上一层父节点中心的方位走向来命名的：NW、NE、SW和SE。八叉树的象限的命名与四叉树类似，便于区分左(L)右(R)、上(U)下(D)以及前(F)后(B)：LUF、LUB、LDF、LDB、RUF、RUB、RDF以及RDB。

除了一些最坏的情形外，可以证明，一个形体用四叉树或八叉树表示时，它的节点数分别与形体的周界或表面成正比[HUNT78；MEAG80]。这一关系是成立的，因为只是从形体的边界表示的需要出发，才增加节点分割。被分割的中间节点只是那些形体的部分边界穿过的节点。因此，在这些数据结构上的任何操作在执行时间上也与形体的周长或面积的大小成正比，数据结构在它包含的节点数目上是线性的。

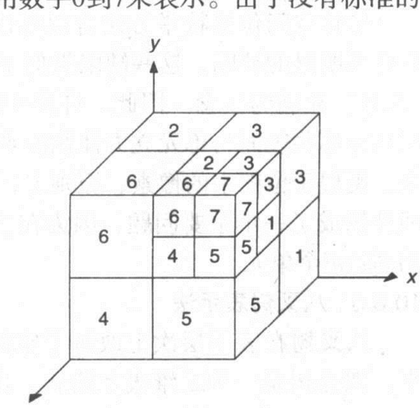


图10-17 八叉树枚举，其中刻度为0的象限不可见

383~384

1. 布尔集合运算和变换

在四叉树和八叉树的存储和处理的有效算法方面已做了许多工作[SAME84；SAME90a；SAME90b]。例如，四叉树和八叉树可以直接进行集合运算 [HUNT79]。为了计算两棵树S和T的交集或并集U，把两棵树自顶向下并列放置。

图10-18表示二棵树的集合运算；这样的运算可直接推广到八叉树。检查每一对相匹配的

节点。考虑集合并的情形。如果一对节点中的一个是黑的，相应的黑的节点加到U中去。如果节点对中有一个是白的，则利用节点对中的另一个节点的值，在U中生成相应节点。如果节点对中两个都是灰色的，则生成一灰色子节点并加到U中，再对这一对节点的子节点递归地运用上述算法。对于第三种情形，运用上述算法后，要检查新节点的孩子节点。如果所有子节点是黑的，则删除它们，并在U中把它们的父节点由灰色变成黑色。

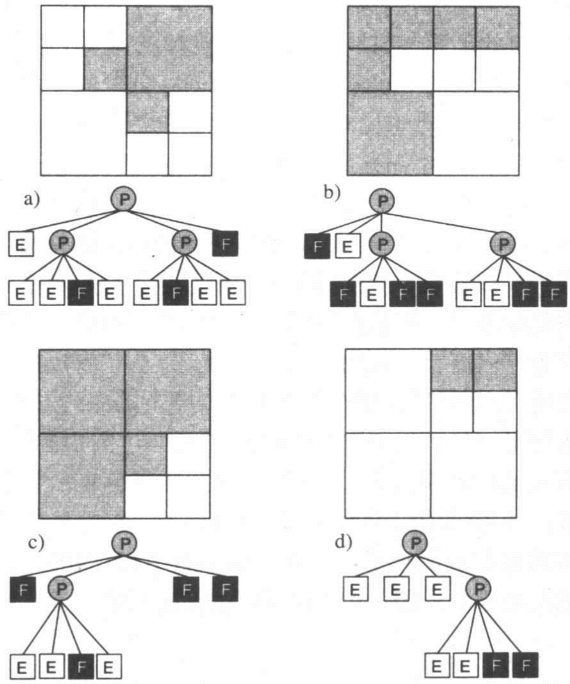


图10-18 实现四叉树的布尔集合运算。a)形体S及其四叉树，
b)形体T及其四叉树，c)$S \cup T$，d)$S \cap T$ 385

在四叉树和八叉树上容易完成简单的变换。例如，绕轴旋转90°的倍数，则对每一层的子节点依次进行同样的旋转。也可以直接进行2次幂倍数的缩放和反射变换。平移变换要稍微复杂一些，对一般的变换也同样如此。另外，与一般的空间位置枚举法一样，在一般变换下，走样问题较为严重。

2. 查找邻节点

四叉树和八叉树的一个重要运算是查找邻节点，即查找与原节点在同一层或者在下层相邻的节点（共享一个面、边或者顶点等）。一个四叉树节点在八个可能方向上有邻节点。它与东、南、西、北方向上的邻节点有共享边，与西北、东北、西南、东南方向上的邻节点有共享顶点。一个八叉树节点在26个可能方向上有邻节点，有共享面的邻节点有6个，共享边的有12个，共享顶点的有8个。

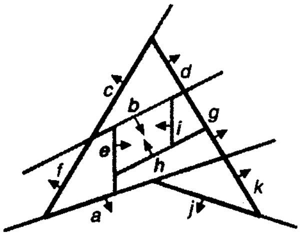
Samet[SAME89a]给出了一种查找指定方向上的邻节点的方法。从原节点开始，上溯四叉树或八叉树，直至找到原节点和邻节点的共同祖先，然后沿树向下找，直至找到所需要的邻节点。这里必须有效解决两个问题：查找共同的祖先，以及决定哪些子孙是它的邻节点。最简单的情形是沿八叉树节点的一个面（L、R、U、D、F或者B）的方向d查找邻节点。从原节点沿树上溯时，共同的祖先是第一个符合下列条件的节点：它不能从原节点d侧的孩子节点到达。

例如，如果要查找它的一个L（左）邻居，则第一个共同祖先是第一个符合下列条件的节点：它不能从孩子节点LUF、LUB、LDF或者LDB到达。因为能从这些孩子节点到达的节点不能有任何在原节点左侧的孩子节点（原节点的左邻居）。当共同祖先找到时，祖先的子树沿着从原节点到祖先的路径的镜像图方向遗传，镜像图是原节点到祖先的路径沿共同边界的反射得到。如果邻节点比原节点大，则只有部分反射路径是相通的。

10.6.4 二元空间划分树

八叉树用三个相互垂直的平面递归地分割空间，在树的每一层上把空间等分成八份。相反，**二元空间划分**（BSP）**树**递归地把空间分成两个子空间，分割的平面可以是任意朝向和任意位置的。二叉树数据结构原来是用于图形学中可见面的判定，有关论述见13.5节。后来，Thibault和Naylor[THIB87]用BSP树来表示任意多面体。BSP树的每一个中间节点对应一个平面，并有两个子节点指针，每一个指向平面的一侧。假如平面的法向指向形体外部，则左边的子节点是指平面的后面或者里面，而右边的子节点是指平面的前面或外面。如果对平面一侧的半空
386 间做进一步分割，则其子节点就成了分割后子树的根节点；如果半空间内部是均匀一致的，则这一半空间的子节点就是叶节点，表示的区域或者全部在多面体的内部或者全部在多面体的外部。这些均一的区域称为“内部”单元和“外部”单元。为了解决运算时限定的数值精度，每一个节点所对应的平面还有“厚度”，所有落在平面误差范围内的点都被看成是在平面上的。

与八叉树和四叉树情形一样，BSP树下的细分概念是与空间维数无关的，因此，图10-19a表示二维空间中用黑线围成的凹多边形。“内部”单元用灰色表示，定义半空间的直线用深灰色表示，法向指向外侧。相应的BSP树如图10-19b所示。在二维空间中，“内部”和“外部”区域把平面分成由凸多边形组成的格子状；在三维空间中，“内部”和“外部”区域把三维空间分成由凸多面体组成的格子状，因此BSP树可以把任意的带“洞”的凹实体表示成凸的“内部”区域的并集。



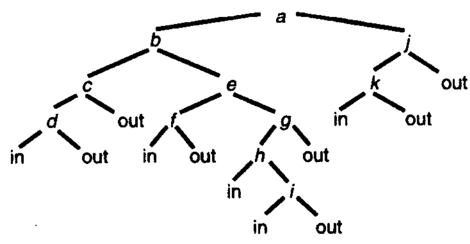


图10-19　二维空间中的BSP树表示。a)凹多边形用黑线作为边界，定义半空间的直线是深灰色的，而“内部”单元是灰色的；b)BSP树

确定一个点是否落在实体的内部、外部或上面这样的问题，称为**点的分类**问题[TILO80]。BSP树可用来对点进行分类，把要分类的点，从根节点开始，向下过滤。在每一个节点，把点代入节点的平面方程，如果点落在节点平面的后面（或里面），递归地通过左子节点，如果点落在平面的前面（或外面），则递归地通过右子节点。如果节点是叶子，则点给出叶子的值，或者“外部”或者“内部”。如果点落在节点平面上，则点通过两个子节点，与上一层的两个分类比较，归到取值相同的一类中；若不同，则点位于“内部”和“外部”区域之间的边界上，分到“上面”类中。这种方法可以推广到对直线和多边形进行分类。但是与点的分类不同，直线或多边形可能部分地位于平面的两侧。所以，在每一节点处，节点平面与直线或者多边形求交，把直线或多边形分割成几个部分，各部分落在平面的前面、后面或者上面，再对各部分单独分类。

Thibault和Naylor描述了从边界表示构造BSP树的算法，为了便于对把BSP树与边界表示结合起来的混合表示进行布尔集合运算，以及确定那些位于BSP树的边界上的多边形[THIB87]。这些算法作用于符合以下条件的BSP树：它的每一个节点与一列嵌入到节点平面 387
内的多边形相对应。利用BSP树的构建算法（将在13.5节中介绍）的改进形式，这些多边形可添加到树中。

虽然BSP树给出了一种简洁的表示，但是在树的构造中，以及进行布尔集合运算时，要对多边形进行分割，从而使得它所表示的记号比其他表示方法潜在地缺少了些紧凑性。但是利用BSP树所固有的与空间维数无关的优点，我们可以研究一种三维BSP树的闭的布尔代数，这种三维BSP树递归地需要把多边形表示成二维树，把边表示成一维树，把点表示成0维树[NAYL90]。

10.7 构造实体几何

在**构造实体几何**（CSG）中，简单的基本实体通过正则集合运算进行组合，其中包括基本实体的直接表示。形体以树的形式存储，集合运算是对中间节点和作为叶子的简单基本实体进行的（图10-20）。一些节点表示布尔算子，而另一些则表示平移、旋转和缩放等操作，很像第7章的层次结构。一般而言，布尔运算不满足交换律，所以树的边是有次序的。

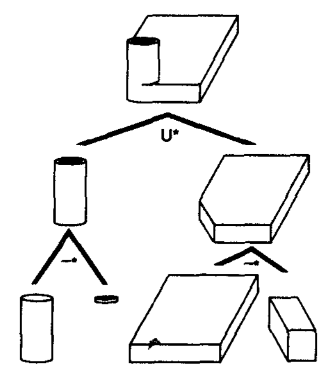


图10-20 由CSG及其树定义的形体 388

为了确定形体的物理性质以及便于绘图，需要把叶子的性质组合起来，以得到根节点的性质。一般的处理策略是第7章所介绍的深度优先遍历树的方法，把叶子节点沿树向上组合起来。这一步骤的复杂度依赖于叶子形体的存储表示方式，以及实际上在树根处的组合形体是否肯定能得到它的完全表示形式。例如，在实现时难以把两个边界表示的节点用正则布尔集合运算（已在10.5节讨论过）组合生成第三个用边界表示的形体。另一方面，通过处理叶子形体的表示而不是显式对它们进行组合，用[FOLE90]中的第15章中讨论的CSG算法可以非常简单地生成一个类似的形体。

在一些实现过程中，基本实体是一些简单的实体，如立方体或球体等，以保证所有的正则组合是有效实体。在另一些系统中，基本实体包括本身并不是有界实体的半空间。例如，一个立方体可定义成六个半空间的交，一个有界圆柱体是无限长圆柱体被顶部和底部的两个平面半空间切割而成。使用半空间会出现生成实体的有效性问题，因为并不是所有的半空间的组合都可生成有效实体。但是，像用平面对形体切片这样的操作可用半空间，要不然需要利用另外一个实体的面参与操作，并且需要引入额外的上限边界，因为在作切片时，即便是只对单个切片感兴趣，也必须是整个实体参与正则布尔集合运算。

我们可以把单元分解法和空间位置枚举法看成是CSG的特殊情形，其中的惟一的操作符是隐式的胶合运算符：两个形体的并，这两个形体可以是接触的，但内部一定不能相交（即形体间的正则布尔交集是空集）。

CSG的表示是不惟一的。如果一个系统允许用户用提拉运算来操作叶子实体，这种表示特别容易使人迷惑。对两个原来相同的形体施加同样的操作，会产生两个不同的结果，如图10-21所示。对CSG模型的编辑可以通过对子树的删除、添加、替换和修改等手段进行，而且CSG模型具有相对紧凑的存储格式，所有这些使得CSG模型成为主导的实体造型表示方法之一。

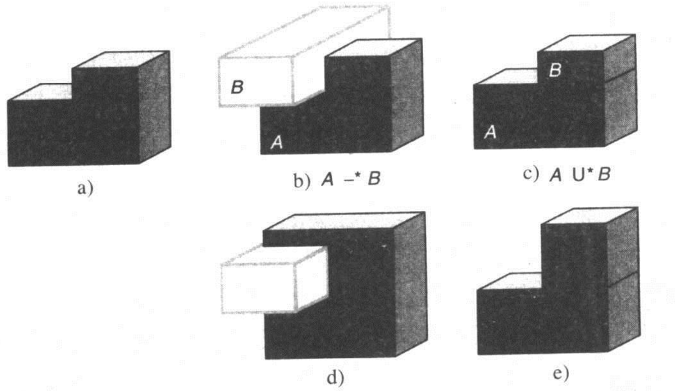


图10-21 图a)所示的形体可以用不同的CSG操作定义，如b)和c)中所示的操作。把b)和c)的两个形体的顶面向上提拉生成不同的形体，如d)和e)中所示

389

10.8 各种表示法的比较

这里已讨论了五种表示：基本实体举例，扫掠表示，边界表示，空间划分表示（包括单元分解法、空间位置枚举法、八叉树法和BSP树法），以及CSG表示法。我们利用10.1节所介绍的准则，对它们进行比较。

- 精确性。空间划分法和多边形边界表示法对许多形体只产生近似表示。但有一些应用，只要近似解是适当的（通常是比较粗糙的），这不算是一个缺点，如寻找机器人的行走路径。但是，要生成满意的视觉图形或者是用足够的精度交互地计算形体，就会产生很大的计算量而不实用。用第14章将要介绍的光滑消隐技术也不能对用多边形构造的人工制品产生栩栩如生的效果。因此支持高质量图形显示的系统经常用包括非多面体基本实体的CSG表示和含有曲面的边界表示。基本实体举例法也可以产生高质量的形体，但不能用集合运算来组合两个简单形体。
- 表示域。能用基本实体举例法和扫掠法表示的形体的域是有限的。相比较而言，空间划分方法可以表示任何实体，尽管通常只是一种近似表示。利用直线边围成的多边形以及其他边和面的类型，边界表示可以用来表示很广泛的形体类型。但是很多边界表示系统仅限于简单的曲面类型和拓扑结构。例如，形体只能是二维流形的二次曲面的组合。
- 惟一性。只有八叉树法和空间位置枚举法才能保证表示的惟一性：只用指定的大小和位置来表示形体。在八叉树的情况下，必须经过一些处理以保证表示方法是完全简化过的（也就是说，没有下面的灰色节点：其子节点都是黑的或白的）。基本实体举例法，一般不能保证表示的惟一性：例如，一个球体可以用球形的和椭球形的基本实体来表示。但是，若精心选取一组基本实体类型，惟一性是可以保证的。
- 有效性。在所有的表示方法中，边界表示最难以保证实体的有效性。不仅是顶点、线、面的数据结构可能不一致，面或边之间也可能相交。相反，BSP树表示一个有效的空间集合，但未必是有界实体。CSG树或八叉树的有效性，只需要进行简单的局部语法检查即可（如果基本实体是有界的，则CSG树表示的实体也是有界的），而空间位置枚举法的

390 有效性则无须任何检查。

- 封闭性。不能用基本实体举例法中的组合方法来构造基本实体，简单的扫掠在布尔运算下是不封闭的。因此在造型系统中，两者通常都不能作为内部表示方式。尽管一些特殊

的边界表示在布尔运算下会碰到封闭性问题（例如，不能表示二维流形以外的边界），但这些情形通常可以避免。

- *紧凑性和效率*。表示方式通常按它们是否生成“已估值的”或“未经估值的”模型分类。未经估值的模型包含了为了完成基本操作必须进一步处理（或估值）的信息，如确定形体的边界等。使用布尔运算，CSG表示生成的是未经估值的模型，因为它使用布尔运算执行的每一次计算都必须沿CSG树遍历，对表达式求值。因此，CSG模型的优点是它的紧凑性以及它记录布尔运算的能力，可以快速地做变换，并可以做快速的撤销（undo）操作，因为这些操作都记录在树的节点上。八叉树和BSP树也可以看成是“未经估值的”模型，由一系列欧拉操作生成的边界表示也可看成是“未经估值的”模型。但另一方面，如果用布尔运算生成形体，它的边界表示和空间位置枚举法通常可看成是“已估值的”模型。需要注意的是这些术语的使用都是相对的，例如，如果所做的操作是确定一点是否在形体内部，则求边界表示的值要比求CSG类似表示的值的工作量要大。

  如第13章所讨论的那样，产生用边界表示和CSG表示的形体的图像有许多有效的算法。虽然对大部分实体而言，空间位置枚举法和八叉树表示只是对形体进行粗略的逼近表示，但是对它们的操纵算法却比对其他表示方式的要简单得多。因此它们已用于由硬件实现的实体造型系统中，这些系统主要应用在对布尔集合运算的速度要求比生成的图像的精度要求更高的情形。

有一些系统使用多重表示方式，因为某种表示方式下的一些运算要比在其他表示方式下的运算效率高。例如，GMSOLID[BOYS82]在存储时使用CSG表示，使得存储更为紧凑，而使用边界表示，以便于对所需数据的快速查询，这些数据在CSG表示中的关系并不明确，如连通性。在GMSOLID系统中，反映形体的当前状态经常是用CSG表示的，只有当需要做进一步运算时，才更新为边界表示。除了有的系统保持两种独立表示方法，并且需要时可以相互转换外，也有些系统是混合表示的，在系统的某一细节层次上，表示方式之间可以相互转化，但系统中只有一种表示信息。有关多重表示和混合表示的一些问题，详见[MILL89]。

正如10.1节所指出的那样，线框表示只包含顶点和边的信息，没有考虑面的信息，这种表 391
示方法本质上是有二义性的。但是Markowsky和Wesley开发了一种算法，它可以由给定的线框表示导出所有可能的多面体[MARK80]，还开发了一种配套算法，由所生成的多面体生成给定的二维投影图[WESL81]。

10.9 实体造型的用户界面

为实体造型系统开发用户界面，可为第8章所介绍界面设计技术的实用化提供了极好的机会。许多操作技术可以在图形界面上进行，包括正则布尔集合运算、提拉以及欧拉算子等的应用，在CSG系统中，允许用户通过修改或替换叶子实体或子树中的节点来编辑形体。表面间的光滑过渡可以用混合和切角两种操作实现。一个成功的系统的用户界面很大程度上依赖于系统内部表示的选择。但是基本实体举例法是个例外，因为它鼓励用户从专用参数方面考虑形体。

在第9章，我们注意到，可以用多种等价方法来表示同一条曲线。例如，画曲线系统的用户界面允许用户通过控制Hermite切向量或者指定Bézier控制点等方式输入曲线，而曲线在系统内部的存储则只是Bézier控制点。类似地，实体造型系统可以让用户用多种不同的表示形式来生成形体，而在系统内部则以另一种形式存储。与曲线表示一样，形体的每一种不同的输入表示都有一些明显的优点，使得这一方式成为生成形体的自然选择。

形体所需要的精度通常是指定几种确定测量精度的方法，例如，通过定位设备或数据录入。

由于一个形体的位置常与另一形体的位置有关，用户界面常常提供两形体间的约束方法。相关的技术是使用户具有定义网格线来约束形体的位置的能力。

在实体造型系统中，用户界面设计的一些最基本的问题起源于在传统的二维交互设备和显示设备上操纵和显示三维形体。这些问题的详细讨论见第8章。许多系统通过提供多个显示窗口来解决其中某些问题，它允许用户同时从不同的位置观察形体。

小结

如上所述，实体造型在CAD/CAM和图形学中都是很重要的。迄今为止，虽然形体的表示
392 和处理手段已有了许多有用的算法和系统，但仍有许多难题需要解决。其中最重要的问题之一就是系统的健壮性。实体造型系统尤其受数值不稳定性的困扰。常用的算法需要实现比硬件更高的精度要求，以保持中间浮点运算的结果。例如，给定两个形体，其中一个形体是另一个形体的副本做非常微小的变形得到的。此时，两形体之间的布尔集合运算就可能失败。

对非刚体、柔性物体、联接体的表示是必要的。许多形体的表示不能指定总的精度，更多的是，形体的形状是由带约束的参数来定义的，参数在一定的范围内取值。这些形体称为“有误差”的形体，对应于由机床和冲床加工出来的真实形体[REQU84]。新的表示方法可用来表示有误差的形体[GOSS88]。

所有已设计的实体有一个共同的、用于一些特殊场合的“特征”，如洞和切边。目前的一个研究领域是探讨自动识别形体的可能性和推断设计者的设计意图，确定每一个特征要完成什么[PRAT84]。这一点允许检查所做的设计以保证特征按设计完成。例如，如果设计某些特征，在压力条件下给定一部分力，则可以自动确认这一功能的实现。也可以检查形体上的进一步操作，以保证不折不扣地实现这些特征。

习题

10.1 在10.2节中定义了完成∩*运算的结果，用同样的方法定义两个多面体形体间的∪*和－*运算的结果。试解释为什么运算结果要限制在正则集，并说明如何确定形体的每个面上的法向。

10.2 考虑确定一个合理的实体是否为空形体（没有体积）的任务。在每一种讨论过的表示方法中，完成这一测试有何困难?

10.3 考虑这样一个系统，其中的形体可以用扫掠表示，并且可以进行正则集合运算。在这样的系统中，对形体要做怎样的限制，才能保证系统的表示是封闭的?

10.4 试解释，在四叉树和八叉树上实现集合运算时，为何不必区分普通的集合运算和在10.2节讨论的正则集合运算。

10.5 虽然作正则集合运算的几何含义是明确的，但是不清楚的是如何对待形体的性质。例如，由两个不同材料构成的形体的交集应指定哪些性质?在制造实际形体时，这一问题并不重要，但对于虚拟的图形世界，任何两种形体之间都可能有交，你认为有什么较好的解决方法?

10.6 试解释如何在图形包中要使用四叉树或八叉树来加快2D或3D拾取。

10.7 描述如何在基本实体举例法、边界表示法、空间位置枚举法以及CSG表示法中实现点的
393 分类。

第11章 消色差光与彩色光

了解光线和颜色的理论和应用，对于现代计算机图形学的学生是十分重要的。即使仅合理使用少许灰度的明暗，也明显增强了绘制对象的外观效果。本书彩图部分展现的许多图像效果，正是使用颜色的结果。颜色是一个很复杂的主题，它涉及物理学、生理学、心理学、艺术和图形设计。在这一章中，我们介绍与计算机图形学关系最紧密的颜色知识。

物体的颜色不仅依赖于物体本身，还依赖于照射它的光源、周围环境的颜色以及人的视觉系统。有些物体（墙、桌子、纸）反射光，有些物体（玻璃纸、玻璃）透射光。如果一个表面仅反射纯的蓝色光，那么在纯红色光的照射下，它呈黑色。类似地，如果一块玻璃仅透射纯红色光，那么如果透过它观察纯绿色光，结果是黑色。这些问题我们将稍后再讨论。我们首先讨论的是消色差光，它被用来描述黑色、灰色和白色。

11.1 消色差光

当我们观看黑白电视或黑白显示器时，我们所观察到的就是消色差光。观察消色差光不会产生红、蓝、黄等感觉，光量是消色差光的惟一属性。如果从物理学中能量的角度讨论光量， 395
可以用术语**亮度**（intensity）和**光强度**（luminance）；如果从心理学中观察到的亮度来描述，可以用术语**辉度**（brightness）。就像我们将要简要介绍的那样，这两种术语代表的含义有关系但并不相同。以一个数值来标识不同的亮度级很有用，定义0代表黑色，1代表白色，介于0、1之间的亮度级代表不同的灰色。

一台黑白电视机在单个像素处能产生多级亮度。行打印机、笔绘图仪和静电绘图仪只能产生两级：纸的白色（或亮灰色）和沉积在纸上墨水或调色剂的黑色（或黑灰色）。后文将要讨论的特定技术可以让只有**两级**亮度的设备产生多级亮度。

11.1.1 选择亮度值

如果我们要显示256种不同的亮度，那么我们应采用哪256级亮度呢？我们选择256这个数目，是由于许多图像中每个像素的亮度由8个二进位来表示。我们当然不希望128个亮度分布在0到0.1之间，另外128个亮度分布在0.9到1.0之间，因为亮度从0.1到0.9是不连续的。开始，我们也许想让亮度级在0与1之间均匀分布，但这种选择忽略了人眼的重要特性：人眼对亮度的比率而非亮度本身敏感。也就是说，我们观察到的亮度0.10与0.11之间的差别和0.50与0.55的差别是一样的。（这种非线性很容易观察到：将灯泡的功率从50瓦调到100瓦再到150瓦，你将看到从50瓦到100瓦的亮度增加比从100瓦到150瓦更大一些）。从辉度（即观察到的亮度）的角度衡量，亮度0.10与0.11之间的差别和0.50与0.55的差别是一样的。因此，为了使辉度均匀分布，亮度级应该呈对数分布而非线性分布。

为了在最低亮度I_0与最高亮度1.0之间寻找256级亮度，每一级亮度是它前一级亮度的r倍，我们使用如下关系式：

$$I_0 = I_0, I_1 = rI_0, I_2 = rI_1 = r^2 I_0, I_3 = rI_2 = r^3 I_0, \dots, I_{255} = r^{255} I_0 = 1 \qquad (11\text{-}1)$$

因此，

$$r = (1/I_0)^{1/255}, I_j = r^j I_0 = (1/I_0)^{j/255} I_0 = I_0^{(255-j)/255} \qquad \text{for } 0 \leqslant j \leqslant 255 \qquad (11\text{-}2)$$

一般地，对$n+1$级亮度，

$$r=(1/I_0)^{1/n}, I_j=I_0^{(n-j)/n} \quad \text{for } 0\leqslant j\leqslant n \tag{11-3}$$

如果亮度只有4级($n=3$)，I_0为1/8（为了说明问题，取的值比实际可能的值要大的多），$r=2$，
396 式(11-3)告诉我们相应的亮度级是1/8，1/4，1/2和1。

一个CRT可能达到的最小亮度值I_0介于最大亮度值的1/200到1/40之间，因此，I_0的典型的值在0.005与0.025之间。最小亮度值不是0，因为CRT的磷涂层是发光的。最大亮度值与最小亮度值之间的比值称为**变化范围**。一个特定CRT的变化范围的准确值可以这样来获得：在一个黑色的区域上显示一个白色的方块，然后用光度计测量这两个亮度。测量必须在完全的黑屋子里进行，避免环境光影响测量结果。取I_0的值为0.02，相应的变化范围为50，根据式(11-2)得到$r=1.0154595\cdots$；并且根据式(11-1)256个亮度中的前几个与后几个分别为0.0200, 0.0203, 0.0206, 0.0209, 0.0213, 0.0216, …, 0.9848, 1.0000。

因为CRT和胶片的非线性，在CRT上正确地显示由式(11-1)定义的亮度不是一个容易的过程，将它们记录在胶片上更加困难。使用一种称为**gamma校正**的技术，可以克服这些困难。gamma校正包括加载光栅显示器的带有补偿值的查找表。过程的细节参见[FOLE90]。

一个自然的问题是“多少个亮度级是足够的？”“足够”指的是数量多的足以显示连续色调的黑白图像。当比率r等于或小于1.01时能达到这种效果（低于这个比率，人眼不能区分亮度I_j与I_{j+1}）[WYSZ82，p.569]。因此，合适的亮度级个数可以通过将$r=1.01$代如式(11-3)得到：

$$r=(1/I_0)^{1/n}\text{，即 } 1.01=(1/I_0)^{1/n} \tag{11-4}$$

解上面的公式得到n为

$$n=\log_{1.01}(1/I_0) \tag{11-5}$$

其中，$1/I_0$是设备的变化范围。

几种显示媒体的变化范围$1/I_0$，以及为了保持$r=1.01$同时充分利用变化范围的亮度级个数n在表11-1中列出，这些都是理论值，前提是具有完美的再现过程。实际上，再现过程中墨水的喷溅和小的随机噪声会大大降低打印媒体的n值。例如，图11-1显示了一个连续色调的照片，图11-2用4和32个亮度级重新表示它。用4个亮度级时，从一个亮度级到下一个亮度级的变化或轮廓非常明显，因为相继亮度级之间的比率r比理想值1.01大出很多。用32个亮度级时，轮廓仅仅能被观察到；而对这个特定的图像，用64个亮度级时，轮廓完全消失。这表明，在纸上打印连续色调的黑白图像（就如本书中的图像）需要的最少亮度级个数是64。然而，对一个处于完全的黑屋子中经过很好调节的CRT，更大的变化范围意味着需要更多的亮度级。

表11-1 变化范围$(1/I_0)$与几种显示媒体所需要的亮度级数量$n=\log_{1.01}(1/I_0)$

显示媒体	典型变化范围	亮度级个数n
CRT	50~200	400~530
像纸	100	465
幻灯片	1000	700
黑白打印方式下的纸	100	465
彩色打印方式下的纸	50	400
黑白打印方式下的报纸	10	234

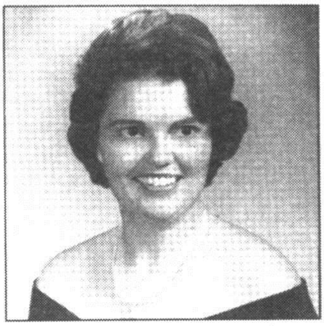
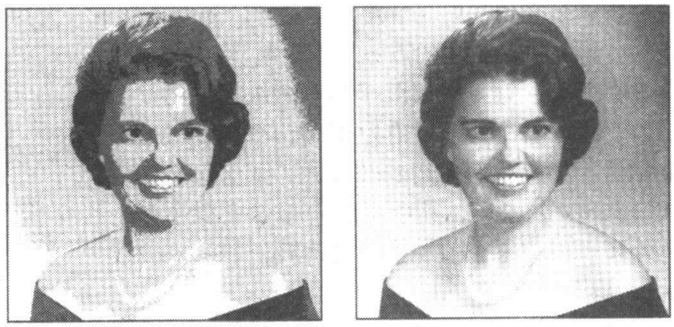


图11-1 一个连续色调的照片



a) b)

图11-2 亮度级对图像再现的影响。a)用4个亮度级表示连续色调的照片，b)用32个亮度级表示连续色调的照片

11.1.2 半色调逼近

许多显示器和硬拷贝设备是单色的，它们只能产生两个亮度级，即使是每像素具有2位或3位的光栅显示器，它所产生的亮度级个数也远少于我们的期望值。我们怎样扩展可用亮度的范围呢？答案在于我们眼睛的**空间综合**能力。如果我们从足够远的距离观察一个很小的区域时，我们的眼睛对区域内的细节进行平均，仅记录区域的总体亮度。



图11-3 半色调在一个小的变化范围内有效地扩展了可用于媒质上的亮度数目。在这个放大的半色调图案中，我们可看到点的尺寸与原照片亮度的变化趋势相反（见图11-1）。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

这种现象被用于报纸、杂志、书上打印黑白照片，这个技术称为**半色调**（在计算机图形学中也称为**聚点有序抖动**）。每一个小的分辨率单位内部由一个黑色的圆形区域填充，该圆形区域的面积与原照片的黑度$1-I$（I是亮度）成正比。图11-3显示了被放大了很多的半色调模式的一部分。注意，这里用水平（称为**屏幕角度**）模式构造倾斜45°的模式。用于报纸的半色调技术每英寸采用60~80个可变尺寸、可变形状的区域[ULIC87]，而用于杂志和书的半色调技术每英寸采用110~200个区域。

图形输出设备能够模拟上述半色调技术中的面积可变的圆形区域。例如，具有两个亮度级的显示器上的一个2×2的像素区域可以用来产生5级亮度，不过这是以降低一倍空间分辨率为代价的。图11-4中的模式可以用来填充2×2的像素区域，其中处于“开”状态的像素个数与需要的亮度成正比。图11-5中的数字化的脸部图像的大小是351×351，它是以2×2模式显示的。 397~399

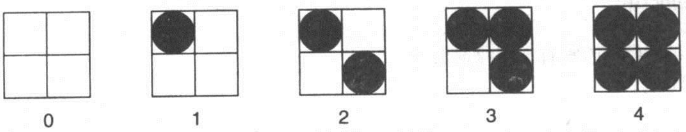


图11-4 用2×2的抖动模式模拟5个亮度级

一个$n\times n$的两级像素区域能产生n^2+1个亮度级。一般地，空间分辨率与亮度分辨率之间有一个折衷。采用3×3的模式将空间分辨率降低为原来的1/3，但它提供了10个亮度级。当然，

这个折衷是受到我们视敏度（在正常的光照条件下，大约1弧分）、观察图像的距离以及图形设备的分辨率（每英寸的点数）的限制。

半色调逼近并不仅限于具有两级亮度的显示器。考虑每个像素具有2位、有4个亮度级的显示器，如果我们采用2×2模式，那么每个模式一共有4个像素，其中的每一个像素除了黑色外还可以取三个值，这就允许我们显示4×3+1=13个亮度。



图11-5 一个连续色调的照片，数字化成351×351大小的图像，用图11-4中2×2模式显示。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

以上介绍的技术假设待显示的图像远小于显示设备的像素阵列，从而可以用多个显示像
400 素表示一个图像像素。如果图像与显示设备的像素阵列大小一样怎么办呢？一种方法是使用由Floyd和Steinberg[FLOY75]开发的**误差扩散**技术，观察结果常常是令人满意的。将误差（像素的准确值与实际显示的逼近值之差）按如下方式加到图像阵列中位于当前像素之右和之下的4个像素值上：误差的7/16加到右端像素，误差的3/16加到左下端像素，误差的5/16加到下端像素，误差的1/16加到右下端像素。这样做的效果是使得误差扩散或混合到图像阵列的多个像素上，图11-6便是采用这种方法创建的。

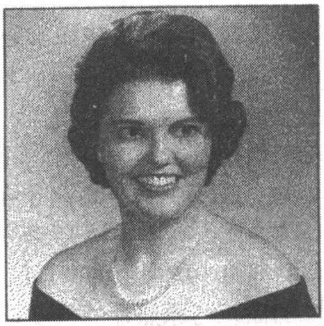


图11-6 采用Floyd-Steinberg误差扩散方法重新生成的具有连续色调的图像。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

给定一个待显示的的图像*S*，要将其在亮度矩阵*I*中显示出来，*S*中被修改的值以及*I*中的显示值按照扫描线的顺序，从最上端的扫描线往下逐行计算：

```
K = Approximate(S[x][y]);      /* 计算与S最接近的可显示亮度值 */
I[x][y] = K;                   /* 显示像素（x, y） */
error = S[x][y] – K;           /* 误差项。必须是float类型的 */

/* 步骤1：将误差的7/16扩散到右端像素(x + 1, y)上 */
S[x + 1][y] += 7 * error /16;

/* 步骤2：将误差的3/16扩散到左下端像素上 */
S[x – 1][y – 1] += 3 * error /16;

/* 步骤3：将误差的5/16扩散到下端像素上 */
S[x][y – 1] += 5 * error /16;

/* 步骤4：将误差的1/16扩散到右下端像素上 */
S[x + 1][y – 1] += error /16;
```
401

为了避免将人工痕迹引入所显示的图像，我们必须保证4个误差的和等于error，不允许存在舍入误差。可以这样来做到这一点：将第4步的误差设为error减去前三步的误差。函数*Approximate*返回与实际像素亮度值最接近的可显示亮度值。对一个具有两级亮度的显示器来

说，*S*的值简单地舍入成0或者1。

如果进一步采用从左向右和从右向左的扫描，可以得到更好的结果。对于从右向左的扫描，第1、2和4步中的从左向右方向上的误差计算反向即可。更详细的讨论和其他误差扩散方法请参见[ULIC87]，其他方法参见[KNUT87]。

11.2 彩色

由彩色光刺激产生的视觉感受要比消色差光丰富得多，讨论彩色通常涉及三个量，它们称为色调（hue）、饱和度（saturation）与明度（lightness）。**色调**区别不同的颜色，如红色、绿色、紫色和黄色。**饱和度**指一种颜色距等亮度的有多远。红色的饱和度高，粉红色的饱和度相对较低。品蓝的饱和度高，天蓝的饱和度相对较低。轻淡的颜色饱和度相对较低，低饱和度的颜色比鲜艳的颜色（饱和度高的颜色）包含了更多的白光。**明度**在非彩色的含义上体现了所观察到的一个反射体的亮度。第四个术语**辉度**（brightness），用来替代明度指观察到的自发光（即发光而不是反光）体的亮度，如灯泡、太阳或CRT。

如果我们要在计算机图形学中精确地应用颜色，就有必要规定和度量颜色。对反射光，我们通过将未知颜色与一个标准的样品颜色集合比较而进行度量。未知颜色与样品颜色必须在标准的光源照射下观察比较，因为观察到的表面颜色同时依赖于表面本身和用于照明的光源。被广泛应用的Munsell颜色排序系统包含了若干组公布了的标准颜色[MUNS76]，这些颜色在三维空间中按照色调、亮度（也就是我们定义的明度）和色浓度（饱和度）来组织。每一个颜色都有名字，它们在颜色空间中是这样排列的：相邻颜色之间的距离相等（由大量的观察者判断得出）。[KELL76]中有关于标准样品颜色、Munsell颜色空间的图表和颜色名称表的进一步讨论。

在打印工业和图形设计领域，颜色一般是通过匹配已打印出的颜色样品来指定的，如由PANTONE MATCHING SYSTEM[PANT91]提供的样品。

画家经常通过高饱和度的或纯色颜料的色浓、色深和色调来区分颜色。向纯色颜料中加白色颜料导致**色浓**（tint）改变，即饱和度降低。向纯色颜料中加黑色颜料导致**色深**（shade）改变，即明度降低。向纯色颜料中同时加黑色颜料和白色颜料导致**色调**（tone）改变。所有这些步骤所产生的颜色的色调相同，它们仅仅改变了颜色的饱和度与明度。仅混合白色颜料与黑色颜料产生灰色。图11-7表示了色浓、色深和色调之间的关系。匹配一个颜色所需的混合颜料的百分比可以用作一种颜色的规范。Ostwald[OSTW31]颜色排序系统与画家的色浓、色深和色调模型相似。

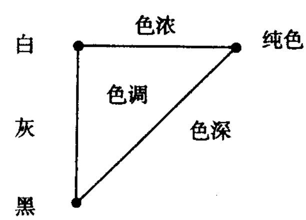


图11-7 色浓、色调与色深

11.2.1 心理物理学

Munsell和画家的颜色混合方法是主观的：它们依赖于观察者的判断、光照条件、样品的尺寸、周围的颜色以及环境的明度。寻找一种客观的、定量的方法是必要的，为此，我们转向物理学的一个分支：**色度学**。色度学中的重要术语有：主波长、色纯度和光强度。

当我们观察一束光时，**主波长**指的是我们所看到的颜色的波长，它与感知意义上的色调相对应。**色纯度**对应颜色的饱和度，**光强度**对应一束光量的多少或光强。一束彩色光的色纯度为纯色光与白色光的比例，纯色光的波长是该彩色光的主波长。完全的纯色光的饱和度是100%，不包含白色光；而纯色光与白色光的混合光的饱和度介于0%与100%之间。白色光和灰色光的饱和度是0%，不含有任何具有主波长的颜色。感知上的术语与色度学术

语的对应关系如下：

感知术语	**色度学术语**
色调	主波长
饱和度	色纯度
明度（反射体）	光强度
辉度（发光体）	光强度

本质上，光是波长介于400 nm到700 nm的电磁波，它们产生的颜色按顺序为靛、蓝、绿、黄、橙和红。在每个波长处光的能量由光谱能量分布$P(\lambda)$表示，如图11-8所示。光谱能量分布包含了无数个数值，每一个数值对应一个可见光谱的波长（实际上，光谱能量分布由光谱中大量的采样点构成，而采样点由分光辐射谱
402~403 仪度量）。幸运的是，我们可以通过一个三元组[主波长，色纯度，光强度]更简洁地描述光谱分布的视觉效果。这就意味着，许多不同的光谱能量分布会产生同样的颜色：它们看起来一样。因此，光谱分布与颜色之间的关系是多对一的。

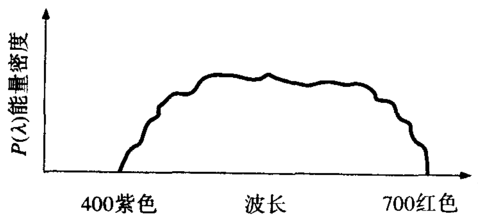


图11-8 一束光的典型光谱能量分布$P(\lambda)$

以上讨论与彩色CRT上红、绿、蓝三色荧光点有什么关系呢？它又与颜色的**三色激励理论**有什么关系呢？三色激励理论基于这样一种假设，即视网膜中有三种颜色感受体（称为视锥体），它们分别对红、绿、蓝三种颜色最敏感。基于这个假设的试验产生的光谱响应函数如图13-18所示。对蓝色的响应峰值在440 nm左右，对绿色的响应峰值在545 nm左右，对红色的响应峰值在580 nm左右（名词"红"和"绿"在这里可能会令人产生误会，因为545 nm和580 nm波长的峰值事实上落在黄色的范围）。曲线表明，人眼对蓝色的响应比它对红色和绿色的响应弱。

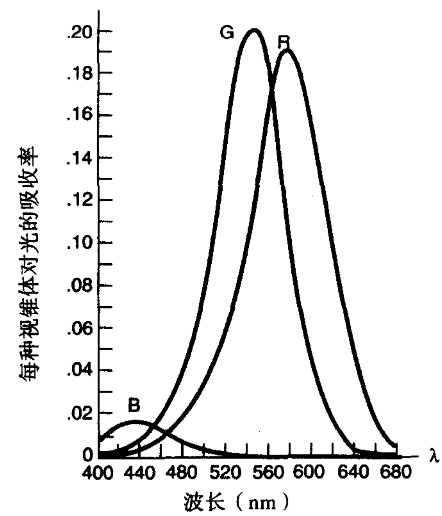


图11-9 人的视网膜中三种锥体的光谱响应函数

图11-10显示了**光效率函数**：随着主波长的变化，人眼对具有恒定光强度光的响应峰值落在波长550 nm左右（黄绿光）。实验证明，这
404 条曲线是图11-9中三条曲线之和。

直观上，三色激励理论具有很大的吸引力，因为它大致与人们关于颜色的一种看法相对应，即一种颜色可以表示为红、绿、蓝（即所谓的基色）正的加权和。这个看法基本上是正确的：图11-11中的三个颜色匹配函数显示了典

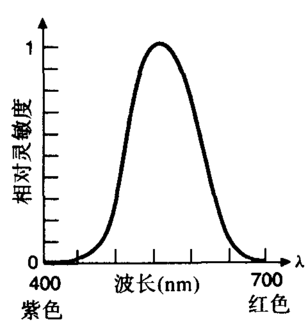


图11-10 人眼的光效率函数

型观察者的观察结果，即对可见光谱内的所有主波长，匹配一种具有恒定亮度的颜色所需要的红、绿、蓝三种颜色光的量。

图11-11中的负值说明，我们不能仅仅通过将基色相加匹配颜色，但是如果将一种基色加到颜色样品中，那么所得到的结果就可以用另外两种基色的混合色进行匹配。因此，图11-11中的负值表明要将基色加到待匹配的颜色中去。颜色匹配时必须取负值并不意味着以下结论是不正确的：将红、绿、蓝混合可以得到其他颜色。相反，用红、绿、蓝混合可以得到大范围内的颜色，否则，彩色CRT就不能正常工作了。但是，它确实说明一些颜色不能够通过红、绿、蓝混合得到，不能在普通的CRT上显示。

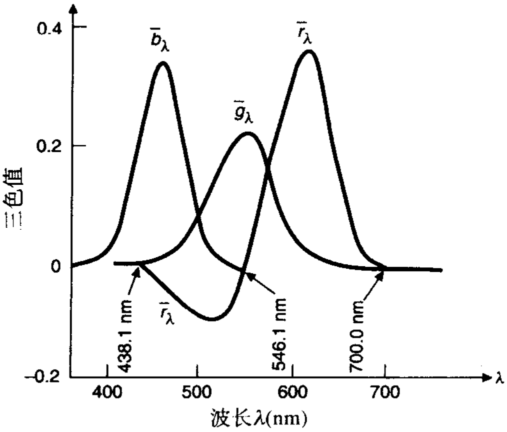


图11-11 颜色匹配函数，显示了匹配可见光谱中所有波长的光所需要的三基色的量

通过将不同的颜色排列在一起，由观察者判断这些颜色是相同的还是不同的，得到结论：人眼能够区分成千上万种颜色。如果两种颜色仅仅是色调不同，那么两个刚好可以区分的颜色波长的差别在光谱的两端超过10 nm，而在480 nm（蓝）和580 nm（黄）左右波长的差别小于2 nm[BEDF58]。除了在光谱的两端，大多数不同色调之间的波长差别在4 nm以内。从而，一共可以区分的满饱和度的色调大约为128种。 405

人眼对饱和度低的光的色调变化敏感度低，而对于固定色调和明度，人眼对饱和度变化的敏感度在可见光谱的两端更大，可区分23个级别。

11.2.2 CIE色度图

用三个固定基色的混合来匹配以致定义颜色是我们所希望的指定颜色的方法，然而图11-11指出，混合时需要负的权值，这很不方便。1931年，国际照明委员会(CIE)定义了三个标准基色，称为**X**，**Y**和**Z**，用以在颜色匹配过程中取代红、绿、蓝。三个相应的颜色匹配函数$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$如图11-12所示。采用这三种基色，只用正的权值就可以匹配我们能看见的所有颜色。基色**Y**有意识地这样来定义，使得它的匹配函数与图11-10中的光效率函数正好一致。注意，就像图11-11中的曲线不是红、绿、蓝的光谱分布一样，$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$也不是颜色**X**，**Y**和**Z**的光谱分布。它们仅仅是一个辅助函数，用来计算需要多少**X**，**Y**和**Z**才能匹配给定的一种颜色。

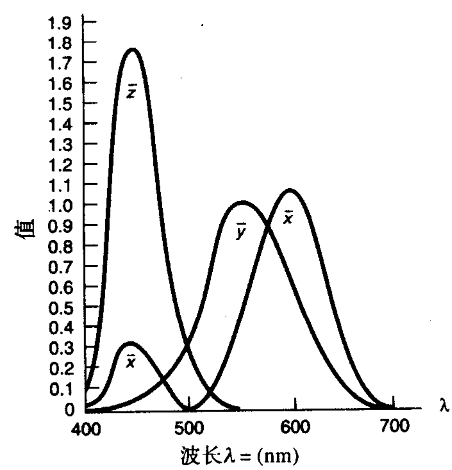


图11-12 1931年CIE三基色**X**、**Y**和**Z**的颜色匹配函数$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$

匹配一个光谱能量分布是$P(\lambda)$的颜色需要的基色**X**、**Y**、**Z**的量为：

$$X = k\int P(\lambda)\bar{x}_\lambda d\lambda,\quad Y = k\int P(\lambda)\bar{y}_\lambda d\lambda,\quad Z = k\int P(\lambda)\bar{z}_\lambda d\lambda \tag{11-6}$$ 406

对自发光体（如CRT）k是680流明/瓦，对反射体，通常选择k值使得白光的Y值为100，从而其他颜色的Y值介于0到100之间。

图11-13显示了XYZ空间中包含所有可见光的锥体，该锥体从坐标原点向正象限延伸，终止于一条光滑曲线。

假设（X, Y, Z）是匹配颜色C需要的CIE三基色的权，如式(11-6)所示，那么，$\mathbf{C} = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$。我们通过规范化（除以$X+Y+Z$）定义色度值（仅依赖于主波长和饱和度，与光的能量多少无关），而$X+Y+Z$可看成总的光能：

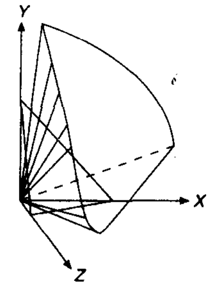


图11-13 CIE颜色空间中可见光形成一个锥体，它从坐标原点向外延伸。其中显示了$X+Y+Z=1$平面。（Cornell大学计算机图形学组，Gary Meyer 1978授权。）

$$x = \frac{X}{(X+Y+Z)},\ y = \frac{Y}{(X+Y+Z)},\ z = \frac{Z}{(X+Y+Z)} \tag{11-7}$$

注意，$x+y+z=1$，也就是说，x、y、z落在图11-13中的平面$(X+Y+Z=1)$之上。彩图4显示了CIE空间和其一部分$X+Y+Z=1$平面，同时也显示了该平面在(X, Y)平面上的正交投影。这个投影即是CIE色度图。

如果我们指定了x和y，那么$z=1-x-y$。但是，从x和y不能够恢复X，Y和Z。为此，我们还需要一个值，通常取Y，它给出了亮度信息。给定(x, y, Y)，从它到(X, Y, Z)的变换为

$$X = \frac{x}{y}Y,\quad Y = Y,\quad Z = \frac{1-x-y}{y}Y \tag{11-8}$$

色度值仅依赖于主波长和饱和度，而与光能多少无关。通过绘制所有可见光的x和y，我们得到CIE色度图，如图11-14所示，它是图11-13平面中的平面$X+Y+Z=1$在(X, Y)平面上的投影。马蹄形区域的内部和边界表示所有可见光的色度值。（色度值相同但亮度不同的颜色对应区域中的同一个点。）光谱上100%纯色落在区域的曲线边界上。用来近似太阳光的标准白光由**光源发光物**C定义，它在区域的中心点标识出来。它的附近某一点（但不是C本身）的色度坐标$x=y=z=1/3$。发光物C的光谱分布近似于色温为6774°K的日光。

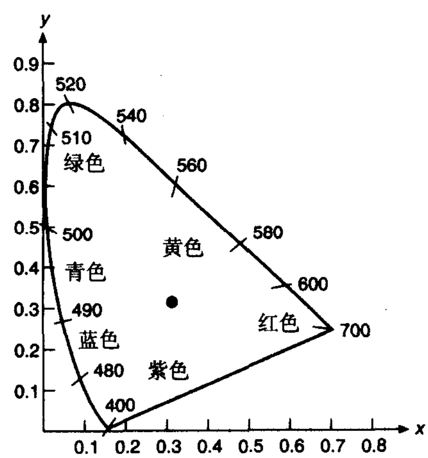


图11-14 CIE色度图。边界上波长的单位是纳米(nm)，黑点标识了发光物C的位置

CIE色度图在很多方面都很有用。首先，通过用CIE三个基色匹配一种颜色，我们可以度量它的主波长和色纯度。现在，假定被匹配的颜色在图11-15中的A点。当两种颜色混合在一起时，在色度图上新产生的颜色位于连接这两种颜色的直线段上。因此，颜色A可看成标准白光（发光物C）和纯色光在B点的混合；从而，B定义了颜色A的主波长，AC与BC的长度之比（表示成百分比的形式）是颜色A的色纯度。A与C越靠近，它包含的白光越多，色纯度越低。

色度图中不包含光强度，从而与光强度相关的色感也被排除在外了。例如，棕色介于橙色和红色之间，但与周围颜色相比，它的光强度非常低，在色度图上没有被显示出来。因此，记住这一点很重要，即色度图不是一个完整的调色板。(X, Y, Z) 空间中有无数张平面，它们的投影都是色度图，在投影过程中它们失去了光强度信息。每一个这样的平面中包含了不同的颜色。

互为**补色**的两种颜色的混合可以产生白光（如图11-15中的D和E）。有些颜色（如图11-15中的F）不能由主波长定义，从而称它们为**非光谱的**。在这种情况下，连接F和C的直线与色度图的马蹄形曲线部分相交于B点，F的主波长就记为其补色主波长后附加c（这里大约是555 nmc），色纯度仍然定义为长度之比（这里是CF比CG）。必须由其补色的主波长的补进行描述的颜色是紫色和洋红色，它们处于色度图的下半部分。

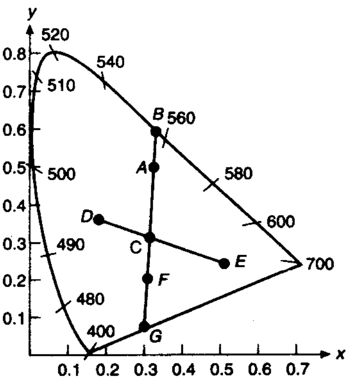


图11-15 色度图中的颜色。颜色A的主波长即为颜色B的主波长。颜色D和E互为补色。颜色F的主波长定义为颜色A的主波长的补

色度图的另一个应用是**颜色域**，它表示了将颜色加在一起的效果。通过调整混合比例，任意两种颜色（比如图11-16中的I和J），加在一起能够产生它们连线上的所有颜色。同样通过调整混合比例，第三种颜色K（见图11-16）与和I、J一起能产生三角形IJK颜色域的所有颜色。色度图的形状表明，可见的红、绿、蓝三种颜色不能通过加法混合来匹配所有的颜色的原因：没有三角形一方面其三个顶点位于可见光区域之内，另一方面又完全覆盖了该区域。

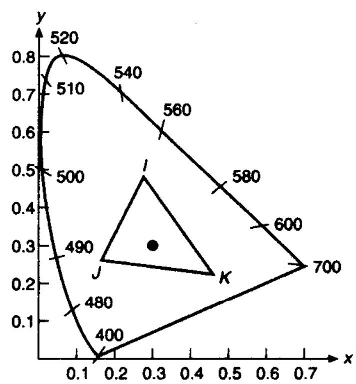


图11-16 混合颜色。混合颜色I和J可以产生直线段IJ上的所有颜色。混合颜色I、J和K可以产生三角形IJK中的所有颜色

色度图也可以用来比较不同彩色显示器和硬拷贝设备的颜色域。彩图15显示了彩色电视机监视器、胶片和打印机的颜色域。与彩色监视器相比，打印机的颜色域较小。为了准确地再现原先在监视器上显示的图像，监视器应该使用精简的颜色域，否则，准确再现是不可能的。但是，如果我们的目标是看起来不错，而不是完全准确再现，那么，颜色域之间的小的差别就不那么重要了。[HALL89]中讨论了如何压缩颜色域的问题。 407~409

有了这些关于颜色的背景知识，我们现在将注意力转向计算机图形学中的颜色。

11.3 用于光栅图形的颜色模型

一个颜色模型定义了一个三维颜色坐标系和该坐标系的一个可见颜色子集，该子集与一个特定的颜色域相对应。例如，RGB颜色模型是三维笛卡儿坐标系中的一个单位立方体子集。

定义一个颜色模型是为了在一个颜色域中方便地指定颜色。我们最感兴趣的是彩色CRT监视器的颜色域，它由RGB（红、绿、蓝）三基色定义，如彩图15所示。在彩图中，我们看到一个颜色域是所有可见颜色的子集，因此，一个颜色模型不能用于指定所有的可见颜色。

三个面向硬件的颜色模型是：用于彩色CRT监视器的RGB模型、用于彩色电视广播系统的YIQ模型和用于一些彩色打印系统的CMY（青色、品红色、黄色）模型。不幸的是，这些颜色模型都不易于使用，因为它们与直观的颜色概念：色调、饱和度、辉度之间并无直接的联系。为此，以易用性为目的开发了另外一类模型，[GSPC79；JOBL78；MEYE80；SMIT78]中介绍了几个这样的颜色模型。这里，我们只介绍一个，即HSV模型（有时也称HSB模型）。

每一个颜色模型都给出了一种向其他模型转换的方法我们将给出在RGB-5 HSV（及CMY）之间和在RGB与YIQ之间的转换方法。[FOLE90]中包含了其他的转换算法。

11.3.1 RGB颜色模型

红、绿、蓝（RGB）颜色模型用于彩色CRT监视器，彩色光栅图形显示系统采用笛卡儿坐标系。RGB基色是**加性**基色，即单个基色的贡献加在一起得到结果颜色，如彩图16所示。我们感兴趣的子集是图11-17中的单位立方体，立方体的主对角线上的颜色包含等量的基色，代表灰色：黑色在(0,0,0)，白色在(1,1,1)。

RGB模型覆盖的颜色域由CRT上荧光物质的色度决定，两个荧光物质不同的CRT具有不同的颜色域。为了将在一个CRT颜色域中指定的颜色转换到另一个CRT的颜色域中，我们可以应用变换M_1和M_2，它们是从RGB颜色空间到（X, Y, Z）颜色空间的变换，详细内容见[FOLE90]。

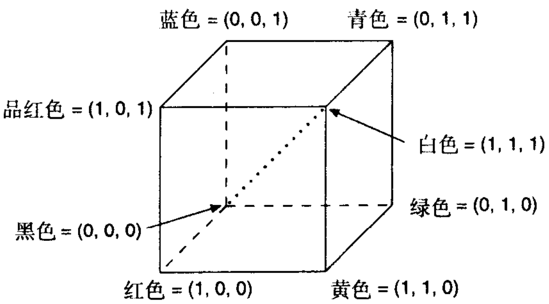


图11-17 RGB立方体，灰色位于主对角点画线上

11.3.2 CMY颜色模型

青色、品红色和黄色分别是红色、绿色和蓝色的补色。当将它们作为滤光片从白光中减去颜色时，它们称为**减性**基色。除了白色（而不是黑色）在坐标原点之外，CMY模型在笛卡儿坐标系中的子集与RGB模型一样。颜色通过如下方式指定：从白光中减去了什么，而不是向黑色中添加了什么。

当处理向纸上添加颜料的硬拷贝设备（如静电或喷墨绘图仪）时，了解关于CMY的知识非常重要。如果一个表面涂上了青色墨水，则它不反射红光，青色从反射的白光（它自己是红色、绿色和蓝色之和）中滤去了红色。因此，从加性基色的角度来说，青色等于白色减去红色，也就是等于蓝色加上绿色。类似地，品红色吸收了绿色，因此它等于红色加上蓝色；黄色吸收蓝色，因此它等于红色加上绿色。一个涂上青色和黄色的表面吸收了红色和蓝色，仅仅让白光中的绿光反射出来。一个涂上青色、黄色和品红色的表面吸收了红色、绿色和蓝色，从而是黑色的。这些关系如图11-18所示，可以从彩图17中看出来，也可以用下面的公式表示：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{11-9}$$

410～411 由1组成的单位列向量是白色的RGB表示和黑色CMY表示。

于是，从RGB到CMY的转换为

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{11-10}$$

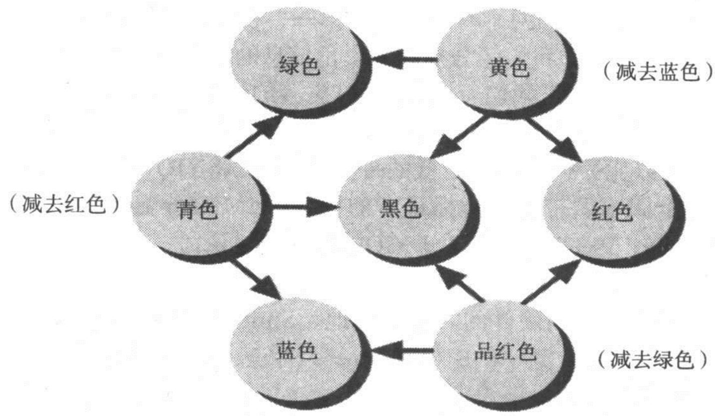


图11-18 减性基色（青色、品红色、黄色）和它们的混合色。例如，青色与黄色混合成绿色

利用这些变换可以将由红、绿、蓝三色的二进制组合得到的八种颜色转换成由青、品红、黄三色的二进制组合得到的八种颜色。这些转换常用于喷墨和静电彩色打印机。

另一个颜色模型CMYK采用黑色（简写为K）作为第四种颜色。CMYK模型用于打印出版和一些硬拷贝设备的四色打印过程。给定一组CMY值，根据下面的关系式，黑色用来取代C、M和Y的等量部分：

$$\begin{aligned} K &= \min(C,M,Y) \\ C &= C - K \\ M &= M - K \\ Y &= Y - K \end{aligned} \tag{11-11}$$

这个主题在[STON88]中有进一步的讨论。

11.3.3 YIQ颜色模型

美国商业彩色电视广播系统采用YIQ模型，因而它与彩色光栅图形密切相关。YIQ是RGB的一种记录方式，目的是进行高效的传输以及向后兼容黑白电视。记录的信号采用NTSC（美国国家电视系统委员会）[PRIT77]系统传输。 412

YIQ模型中的Y代表的不是黄色，而是光强度，它的定义与CIE的Y基色相同。黑白电视只显示彩色电视信号中的Y分量：色度信息包含在I和Q信号中。YIQ模型采用三维笛卡儿坐标系，可见颜色子集构成一个凸多面体，它映射为RGB立方体。

RGB到YIQ的映射定义如下：

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{11-12}$$

第一行的值反映出绿色和红色对辉度的贡献大，而蓝色对辉度的贡献相对较小。以上矩阵的逆矩阵完成从YIQ到RGB的转换。

用YIQ模型指定颜色带来了一个潜在的因广播电视材料引起的问题：两个不同的颜色在一个彩色监视器上并排显示，看起来不一样；但是，如果转换成YIQ并在黑白电视上显示，它们看起来就一样了。这个问题可以这样来解决，即在YIQ颜色模型空间中调整这两种颜色（仅调整Y值），使它们的Y值不一样。

YIQ模型利用了我们视觉系统的两个有用特性。第一，视觉系统对光强度的变化比对色调和饱和度的变化更敏感，也就是，我们区分空间中彩色信息的能力弱于区分空间中单色信息的能力。这表明，在表示Y、I和Q时，我们应该分配给Y更多的带宽，使得Y的分辨率高一些。第二，占据我们视场很小一部分的物体只能产生有限的色感，采用一个颜色维数就足以表示它了。这表明，I和Q中的一个可以比另一个拥有更低的带宽。NTSC将YIQ调制编码到一个广播信号中，利用以上特性在这个固定的带宽上传播最多的信息：4MHz分配给Y，1.5MHz分配给I，0.6MHz分配给Q。[SMIT78；PRIT77]中有关于YIQ的进一步讨论。

11.3.4 HSV颜色模型

RGB、CMY和YIQ模型是面向硬件的。相对应地，Smith的HSV（色调、饱和度、亮度）模型[SMIT78]（也称为HSB模型，其中的B（brightness）代表辉度）是面向用户的，它基于画家的颜色概念：色浓、色深与色调。该模型采用的坐标系是圆柱形坐标系，模型本身定义为其中的一个**六棱锥**或六面金字塔，如图11-19所示。六棱锥的顶面对应$V=1$，其中包含了较亮的颜色，但是，$V=1$平面中的颜色也并不是同样的辉度。

色调（即H）由绕纵轴的角度度量，红色对应0°，绿色对于120°，依此类推（见图11-19）。在HSV模型中，互补的颜色相差180°。S的值从0到1变化，它位于中心线（V轴）上时值为0，位于三角边上时值为1。饱和度的度量是相对于该颜色模型所代表的颜色域的，而后者当然是整个CIE色度图的一个子集，因此，模型中饱和度为100%的颜色其色纯度小于100%。

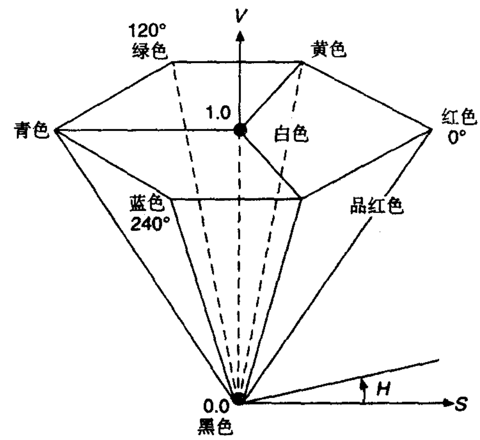


图11-19 单六棱锥HSV颜色模型。$V=1$平面包含了所示区域中的RGB模型的$R=1$、$G=1$和$B=1$平面

六棱锥的V向高度是一个单位，顶点位于坐标原点。顶点对应黑色，V坐标是0。在该点处，H和S的值没有定义。对应$S=0$、$V=1$的点是白色，对应$S=0$而V取中间值的点是灰色。当$S=0$时，H的值没有定义。当$S\neq 0$时，H可有相应值。例如，红色对应$H=0$、$S=1$、$V=1$。事实上，对应$V=1$、$S=1$的任何颜色与画家所用的纯色颜料（用来混合产生其他颜色）都相近。添加白色颜料相当于降低S（不改变V），保持$S=1$降低V值改变色深，同时降低S和V值改变色调。当然，改变H相当于选择不同的纯色颜料。因此，H、S和V与画家颜色系统的概念密切对应，而与11.2节引入的术语不完全类似。

如果沿着主对角线从白色点到黑色点对RGB颜色立方体进行投影，所得到的投影结果与HSV六棱锥的顶面相对应，如图11-20所示。RGB立方体有子立方体，如图11-21所示。同样沿着主对角线观察，每一个子立方体也与图11-20中的六边形一样，只是小一点。在HSV空间中固定V值所得到的每个平面就对应着这样的RGB空间中子立方体的视图。从而，我们直观地建立了RGB模型与HSV模型的对应关系。程序11-1和程序11-2中的算

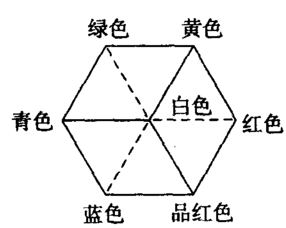


图11-20 沿着主对角线观察RGB颜色模型，立方体的可见边是实线，不可见边是虚线

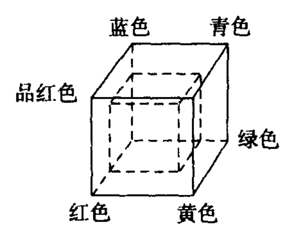


图11-21 RGB立方体和子立方体

法精确地给出了从一个模型到另一个模型的转换。

程序11-1 从RGB颜色空间到HSV颜色空间的转换算法

```
void  RGB_To_HSV(float r, float g, float b, float *h, float *s, float *v)
{
/* 给定：r，g，b属于[0,1] */
/* 期望：h属于[0,360 ], s,v属于[0,1]，或者如果s = 0，那么h = UNDEFINED */
/* （没定义）,它是超出[0，360]的值定义的常量* /
   float max, min, delta;

   max = MAX(r, g, b);
   min = MIN(r, g, b);
   *v = max;                  /* 这是v值 */
   /* 下面计算饱和度s */
   if (max != 0.0)
      *s = (max – min) / max;    /* S是饱和度 */
   else
      *s = 0.0;               /* 如果红、绿、蓝的值都是0，饱和度是 0 */
   if (*s == 0.0) {
      *h = UNDEFINED;
      return;
   }
   /* 彩色的情况：饱和度不是0，因而确定色调 */
   delta = max – min;
   if (r == max)            /* 结果颜色在黄色与品红色之间 */
      *h = (g – b) / delta;     /* 结果颜色在青色与黄色之间 */
   else if (g == max)
      *h = 2.0 + (b – r) / delta;  /* 结果颜色在品红色与青色之间 */
   else if (b == max)
      *h = 4.0 + (r – g) / delta;
   *h *= 60.0;               /* 将色调值转换成度数 */
   if (*h < 0.0)
      *h += 360.0;            /* 确保色调值是非负的 */
}
```

程序11-2 从HSV颜色空间到RGB颜色空间的转换算法

```
void  HSV_To_RGB(float *r, float *g, float *b, float h, float s, float v)
{
/* 给定：h属于[0,360 ]或者UNDEFINED，s,v属于[0,1] */
/* 期望：r, g, b属于[0, 1] */
   float f, p, q, t;
   int  i;

   if (s == 0.0) {                /* 颜色在黑白中心线上 */
      if (h != UNDEFINED) {       /* 非彩色：没有色调 */
         Error();               /* 按照规定，如果s = 0并且h有值，则出错 */
         return;
      }
      *r = v;
      *g = v;
      *b = v;
      return;
   }
                                  /* 彩色：s≠0，从而有色调 */
   if (h == 360.0)                /* 360度与0度等价 */
      h = 0.0;
```

413
～
415

```
    h /= 60.0;                  /* h属于[0, 6] */
    i = floor(h);               /* floor返回小于等于h的最大整数 */
    f = h - i;                  /* f是h的小数部分 */
    p = v * (1 - s);
    q = v * (1 - s * f);
    t = v * (1 - s * (1 - f));

    switch (i) {
    case 0:
        *r = v;
        *g = t;
        *b = p;
        break;

    case 1:
        *r = q;
        *g = v;
        *b = p;
        break;

    case 2:
        *r = p;
        *g = v;
        *b = t;
        break;

    case 3:
        *r = p;
        *g = q;
        *b = v;
        break;

    case 4:
        *r = t;
        *g = p;
        *b = v;
        break;

    case 5:
        *r = v;
        *g = p;
        *b = q;
        break;
    }
}
```

416

11.3.5 颜色的交互指定

许多应用程序允许用户指定区域、直线段、文本等的颜色。如果系统只提供少数几种颜色，将可用的样品颜色在菜单列出来以供选择是可行的；但如果颜色多得无法合适地列出来，怎么办呢？

简单的办法是用英语名称（直接输入或者用滑动刻度盘输入）在颜色空间中指定颜色的数值坐标，或者直接与颜色空间的图形表示进行交互。命名方法通常不能令人满意，因为名字的含义模糊并带有主观性（“淡海蓝色外加一点绿色”），这与图形交互技术的目标是相对立的。但[BERK82]中描述了一种定义良好的颜色命名机制——CNS，它用 “带绿色的黄色”、“绿黄色”和 “带黄色的绿色” 等术语区别介于绿色与黄色之间的三种色调。在一个试验中，一组用户通

过CNS指定颜色，另一组在RGB或HSV空间中输入数值坐标指定颜色，结果是前一组比后一组更准确。

在任一种颜色模型中指定颜色坐标可用滑动刻度板实现。如果用户了解其中的每一个值对颜色的影响，这项技术则非常有效。也许最好的颜色交互指定方法是与颜色空间的图形表示直接交互，如图11-22所示。在圆形区域（代表$V=1$平面）内转动直线决定了HSV立体中哪一个剖面在三角形区域内显示，三角形区域内的光标可以移动用来确定饱和度与亮度值。当直线或光标移动时，数字读数器的值发生改变。当用户在读数器中直接键入新的数值时，直线段和光标的位置也随着发生变化。颜色样品框显示当前被选中的颜色。然而，一个人对颜色的观察结果受周围颜色以及颜色区域大小的影响，因此，在反馈区域中观察到的颜色也许与实际观察结果不一致。所以，用户在设置颜色时，也要同时观察一下实际的显示结果。

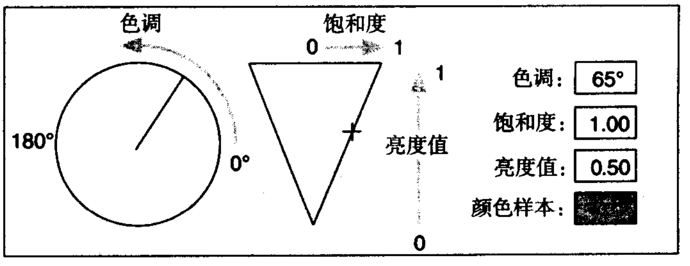


图11-22 一种在HSV空间中指定颜色的方便的方法。饱和度和亮度由光标在三角形区域内指示，色调由圆形区域内的直线段指示。用户可以移动图中的直线段和光标指示器，读数器中的数值将随着更新。另一方面，用户也可以键入新的数值，那么指示器也随着更新位置。也可以为H、S和V分别增加滑动刻度板，让用户一次在一个维数上精确控制它们的数值，而不再需要键入 417

11.3.6 在颜色空间中进行插值

颜色插值至少在三种情况下是必须的：Gouraud明暗处理（14.2.4节）、反走样（3.14节）和对两个图像进行溶合，如淡入淡出序列。颜色插值的结果依赖于我们在哪个颜色模型中进行插值，因此，必须要小心地选择一个合适的颜色模型。

如果从一个颜色模型到另一个颜色模型的转换将直线段（表示插值路径）变为直线段，那么在这两个模型中进行插值的结果是一样的。RGB、CMY、YIQ和CIE颜色模型就属于这种情况，它们之间通过一个简单的仿射变换联系起来。然而，RGB模型中的一条直线段通常不能转换到HSV模型中的一条直线段。彩图19显示了在HSV、RGB和YIQ颜色空间中对两个相同的颜色进行线性插值的结果。考虑在红色和绿色之间插值。在RGB模型中，红色=(1, 0, 0)，绿色=(0, 1, 0)，它们的插值（为了方便取权值都等于0.5）是(0.5, 0.5, 0)。将算法RGB_To_HSV（程序11-1）应用于这个结果，我们得到(60°, 1, 0.5)。现在，在HSV模型中表示红色和绿色，我们有(0°, 1, 1) 和(120°, 1, 1)，用同样的权值在HSV模型对它们插值，得到(60°, 1, 1)，结果与在RGB模型中插值相差0.5。

作为第二个例子，考虑在RGB模型和HSV模型中对红色和青色插值。在RGB模型中，我们从（1, 0, 0）和（0, 1, 1）开始，插值得到（0.5, 0.5, 0.5），而这个结果颜色在HSV模型中表示为(UNDEFINED, 0, 0.5)。在HSV模型中，红色和青色分别是(0°, 1, 1)和(180°, 1, 1)，插值结果得到(90°, 1, 1)；亮度和饱和度保持最大时得到了一个新的色调，而“正确”的结果是：两个等量的补色混合得到的应该是灰色。这又一次表明，插值然后变换与变换然后插值所得到的结果是不同的。

Gouraud明暗处理方法可以采用任一种颜色模型，因为两个待插值的颜色非常靠近，从而插值路径也非常靠近。当两个图像做溶合时，如在淡入淡出序列和反走样中，颜色有可能相距很远，此时采用加性模型（如RGB）比较合适。另一方面，如果目标是在两个具有固定色调（或饱和度）的颜色之间进行插值，并且希望保持所有颜色的色调（或饱和度），那么采用HSV模型更好。

11.4 在计算机图形学中应用颜色

我们应用颜色的目的是为了美，为了建立一种氛围和情绪，为了真实，为了强调，为了识别相关联的区域，或者为了进行编码。通过仔细的考虑，颜色可以被有效地用来实现这些目标。
418 另外，用户喜欢颜色，尽管还没有数据证明颜色能提高他们的效率。

粗心地应用颜色可能会使显示器比一个相应的单色演示可用性更差、更不具吸引力。在一个实验中，引入了无意义的颜色使用户性能减少到引入之前的三分之一[KREB79]。对颜色的应用应该适度，任何装饰性的使用应服从于功能性使用，使颜色不会被误解为具有什么潜在的含义。因此，就像人机界面的其他方面一样，颜色的使用必须由实际用户来测试，以便发现和解决问题。当然，有些人可能有其他偏爱，因此，通常的做法是，基于颜色使用规则提供缺省的选择，同时给用户提供改变缺省值的手段。一个谨慎的颜色选择方法是，首先为单色显示器设计，保证颜色使用是完全冗余的。这避免了给没有彩色设备的用户带来问题，也意味着这个应用可以在单色显示器上应用。在彩图9和彩图10显示的窗口管理器中，颜色选项设计是相当谨慎的。颜色不是按钮状态、选中的菜单项等的惟一编码。

许多书讨论了颜色在美学上的应用，包括[BIRR61]，我们在这里仅介绍有助于产生和谐颜色的几条简单规则。颜色美学最基本的规则是根据一些方法选择颜色，通常，在颜色模型中沿着一条光滑的路径遍历颜色，或将颜色限制在颜色空间内的平面或六棱锥上。这也许意味着选择具有恒定明度的颜色。此外，颜色最好在观感上是均匀分布的（这与在某个坐标方向上等间隔分布是不一样的，实现起来很困难）。注意，在不同的颜色空间中对两个颜色进行线性插值（如在Gouraud明暗处理方法中）得到的结果不同（参见习题11.6和彩图19）。

随机地选择色调和饱和度通常显得有些过分艳丽。Alvy Ray Smith做了一个非正式的实验，在这个实验中16×16个格子用随机产生的颜色分别填充，得到的结果并不意外：格子毫无吸引力。根据H、S和V值对256种颜色进行排序并按照新的顺序重新显示，格子看起来明显改善了。

关于这些规则的更多具体的例子表明，如果一个图表中仅包含几种颜色，背景应该选用其中一种颜色的补色。如果一幅图中包含许多颜色，背景应该选用一种中性的颜色（灰色），因为它既和谐又不显眼。如果两个相邻的颜色不是特别和谐，可以用细的黑色边框将它们分开。边框的应用在非彩色（黑/白）视觉通道中更有效，因为黑色的轮廓能加速形状检测。以上讨论的一些规则在ACE（A Color Expert）中被编码，ACE是一个用于选择人机界面颜色的专家系统[MEIE88]。一般地，尽量少用不同的颜色总是好的（真实感图形的明暗处理除外）。

颜色可以用于对信息进行编码，如彩图20显示的那样。但是，按照次序有几点需要注意，第一，颜色代码很容易携带并不想要的含义。将A公司的收入显示成红色，而B公司的收入显示成绿
419 色可能足以使人理解为：A公司出现了经济困难，因为我们通常总是将颜色与一些含义联系在一起的。亮的、饱和度高的颜色比暗淡的、灰白的颜色更突出，可能会强调并不存在的重点。显示器上两个具有同样颜色的元素可能因为相同的颜色代码而被视为是相关联的，虽然实际并非如此。

当颜色同时被用来对菜单项分组和区分图形元素如印制电路板的不同层或VLSI芯片时，这个问题经常出现。例如，我们倾向于把绿色的图形元素与同样颜色的菜单项关联起来。这是

限制应用于人机界面元素（如菜单项、对话框和窗口边界）的颜色的原因之一。（另一个原因是将尽量多的颜色留给应用程序自己。）

许多颜色使用规则是基于生理学而不是美学的考虑。例如，因为人眼对亮度的空间变化比对色调的空间变化更敏感，线段、文本以及其他的细节图形不仅要在色调上而且要在辉度（观察到的亮度）上与背景相区别，特别是对那些包含蓝色分量的颜色，因为只有相对较少的视锥体对蓝色敏感。从而，两个仅仅在蓝色分量上有所区别的等亮度彩色区域的边界很难辨别出来。另一方面，对蓝色敏感的视锥体比对红色和绿色敏感的视锥体分布得更广，因而我们的外围彩色视觉对蓝色更好，这解释了为什么许多警车的闪光灯现在采用蓝色而不是红色。

蓝色与黑色在辉度上相差甚小，它们是一对特别差的组合。类似地，黄色在白色上很难区分，因为这两种颜色都很亮。彩图10显示了在白色的背景上，利用黄色非常有效地加亮了黑色文本。黄色与黑色文本的对比度很大，并且也很突出。另外，应用于文本反色时，用黄色加亮并不比用黑色加亮（也就是说，被加亮的文本是白色，文本背景是黑色，这在单色显示器上很常见）更好。

蓝色背景上的白色文本提供了很好的对比度，同时它也不像白色在黑色上那么刺眼。最好避免低饱和度和低光强度的红色和蓝色，因为红绿色盲者（最常见的色觉缺陷）不能区分这些颜色。Meyer和Greenberg描述了如何为色盲观察者选择颜色[MEYE88]。

眼睛不能区分非常小的物体的颜色，就像在介绍YIQ NTSC颜色模型时已经提到的那样，因此，颜色编码不应该用于小的物体。特别地，判断对着20到40弧分（1弧分 = 1/60度）的物体的颜色容易出错[BISH60；HAEU76]。一个高0.1英寸的物体，如果从24英寸（一个典型的观察距离）处观察，它占据这么大的弧度数；如果15英寸显示器纵向上有1024条扫描线，那么它对应7个像素高。很明显，单个像素的颜色是很难辨别的（参见习题11.10）。

观察一个区域所感受到的颜色受周围区域的颜色影响，如果用颜色对信息编码，这种影响就特别成问题。当周围区域的颜色是灰色或相对不饱和的颜色时，这种影响达到最小。

区域的颜色实际上能够影响它的观察尺寸。Cleveland和McGill[CLEV83]发现，对于一个 420
红色方块和一个绿色方块，看起来，红色方块更大。这种效果足以使观察者认为红色方块比绿色方块更重要。

如果一个用户盯着颜色饱和度很高的一块大区域几秒钟，然后看往别处，这块大区域的残留图像就会出现。这种影响令人不安，并且使眼睛疲劳。因而，使用高饱和度颜色的大区域是不明智的。同样，不同颜色的大区域看起来与观察者距离不一样，因为光的折射依赖于波长。当观察者从凝视一种颜色的区域改为凝视另一种颜色的区域时，眼睛改变了聚焦点，这种聚焦的改变形成了深度不同的印象。红色和蓝色，在光谱的两端，具有最强的深度不一致效果，红色看起来更近，蓝色看起来更远。因此，同时将蓝色用于前景物体而红色用于背景是不明智的，反过来则很好。

了解了关于颜色使用的所有这些危险和缺陷，你对我们把谨慎地使用颜色作为第一个规则还感到惊奇吗?

小结

随着彩色监视器和彩色硬拷贝设备在许多应用中变成标准配置，颜色在计算机图形学中也越来越重要。在这一章中，我们介绍了那些与计算机图形学最相关的颜色概念，如果想了解这方面的更多信息，请查阅关于颜色的大量文献，如[BILL81；BOYN79；GREG66；HUNT87；JUDD75；WYSZ82]。更多的关于在计算机图形学中应用颜色的艺术和美学方面的问题请参见[FROM84；MARC82；MEIE88；MURC85]。关于如何精确调节监视器以及如何匹配监视器颜

色与打印颜色的问题在[COWA83；STON88]中有所讨论。

习题

11.1 计算每个像素有w位，大小为$m \times m$的像素模式能表示的亮度数目。

11.2 写一个算法，在具有两级亮度的输出设备上显示像素阵列，算法的输入是每个像素有w位$m \times m$像素亮度阵列和一个$n \times n$的增序矩阵，即矩阵中任意一个像素亮度增强到j级，那么该矩阵中所有亮度级为$k > j$的像素也增强。假定输出设备的分辨率是$m \cdot n \times m \cdot n$。

11.3 写一个算法，用$n \times n$填充模式在只有两级亮度的设备上显示一个填充多边形。

11.4 当用一定的模式填充显示在隔行扫描显示器上的多边形时，所有处于“开”状态的位要么落在奇数行扫描线上要么落在偶数行扫描线上，引起了一定的闪烁。修改习题11.3中的算法，交换$n \times n$模式的各行使得模式的各行能够交替利用奇数和偶数行扫描线。图11-23
421 显示了对图11-4采用亮度1得到的结果，a)图进行了交替，b)图没有。



a) b)

图11-23 用两种方法对图11-4取亮度1得到的结果：a)交替（加亮的像素同时落在两条扫描线上），b)非交替（所有加亮的像素落在同一条扫描线上）

11.5 在RGB立方体和HSV六棱锥上标出亮度值分别是0.25、0.50和0.75的点的位置，其中亮度值由$Y = 0.299R + 0.587G + 0.114B$定义。

11.6 用R、G、B表示YIQ模型中的I和HSV模型中的V，注意I和V是不一样的。

11.7 讨论一个光栅显示器的设计，如果它采用HSV模型（而不是RGB模型）来表示颜色。

11.8 重写HSV到RGB的转换算法，使它的效率更高。用表达式$vs = v*s$；$vsf = vs*f$；$p = v - vs$；$q = v - vsf$；$t = p + vsf$替换p, q和t的赋值表达式。假定R、G、B在区间[0，255]中，看看有多少运算可以转换成整数运算。

11.9 写一个程序，在显示器中并列显示两个16×16网格，用颜色填充它们。左边的网格包含从HSV颜色空间中随机选出的256种颜色（对H、S、V，用随机数发生器从10个等间隔的数值中选择一个）。右边的网格包含同样的256种颜色，但是按照H、S、V的值做了排序，变换排序关键字H、S、V，观察实验结果。

11.10 写一个程序，在一个灰色背景上显示橙色、红色、绿色、蓝色、青色、品红色和黄色小方块，每一个方块与其他方块是相互分离的，大小为$n \times n$像素，n是一个输入的变量。为了在24英寸和48英寸的距离处清楚地判断每个方块的颜色，n的值必须为多大？这两个n值之间有什么关系？不同的背景色对这个结果有什么影响？

11.11 给定亮度变化范围是50、100和200，计算为了保存256个亮度级所需要的查色表每单元的位数。

11.12 写一个程序，在RGB和HSV模型中对两个颜色线性插值。接受两个颜色作为输入，并允
422 许在这三个模型的任一个中指定它们。

第12章 可视图像真实感的探讨

前几章我们讨论了简单的二维和三维图元的图形技术。我们所产生的图片（例如，第6章中房屋的线条图）表示了在现实生活中其结构和外观上都复杂得多的物体。本章中，我们介绍一种越来越重要的计算机图形学的应用：生成三维场景的真实感图像。

何为真实感图像？由绘画、拍照或是计算机所产生的场景图片，在什么意义上可以说是“真实的”是一个很有争议的学术题目[HAGE86]。我们较为广泛地使用这一名词，用以指那些体现了光线与真实物体相互作用所具有的许多效果的照片。这样，我们将真实感图像看成图片及其生成技术的一种引申和不严格的说法，也就是说，看成“多”或“少”真实感。在延伸的一端的例子是通常被称为**照相真实感**（或**真实感照相**）。这些图像试图综合聚焦在对着所描述物的照相机胶片底版上的光照强度的区域。按照我们的方法，延伸的另一端，我们将寻觅能持续地提供较低的可视线索的图像。

在思想上应容忍这样的观点，那就是：一张更有真实感的图片不必是更理想的或更有用的图片。如果一张图片的最终目的是传达信息，那么，一张没有复杂的阴影和反射的图片也许比一张真实感照片的绝技更成功。另外，在随后章节中，概要描述该技术的多种应用，真实性被有意地改变成了审美的效果或满足天真的观察者的期望。同样的道理，科幻影片的特性可以做到在外层空间中武器的爆炸声——在真空中这是不可能的。例如，在彩图23中显示的天王星，Blinn在行星黑暗面加入了额外的光线，形成反差，以使得所有的特征是同时可见的，否则，星球的黑暗面将是黑的。在物理上的随意性可产生具有魅力的、令人难忘的有用的图片。 423

生成真实感图片涉及若干步骤，这些步骤将在随后的章节中详述。虽然，这些步骤往往视为构成了概念流水线，但正如我们将看到的，根据所用的算法，这些步骤的实现顺序可以不同。首先，用第9章和第10章中所讨论的方法来生成对象的模型。其次，选择一个观察规范（如第6章中所开发的）和光照条件。由第13章所讨论的算法来决定观察者的可见面。在可见面的投影中，对每个像素所赋的颜色是对象所反射和透射的光线的一个函数，并由第14章所论述的方法确定。然后，最终的结果照片可使用合成技术，与早先所生成的那些照片组合而成（例如，再使用一个复杂的背景）。最后，若我们正产生一个动画序列，在造型、光照以及观察者所做的规定等随时间所做的改变，必须加以定义。从模型产生图像的过程通常被称为**绘制**。**光栅化**这个术语是专门指从输入几何图元到确定像素值的各个步骤。

本章从多个角度提出真实感绘制问题。首先，我们注意一些已经使用真实感图像的应用。然后，我们概略地回顾历史进程，考察了能成功地生成较真实的图片的一系列技术。每种技术都以一张用到其新技术的标准场景的图片来说明。最后，我们以如何进入后续各章的建议结束本章。

12.1 为什么讨论真实感

产生真实感的图片是模拟、设计、娱乐和广告、科研和教育以及指挥和控制等领域中的一项重要目标。

模拟系统出示的图像，不仅要有真实感，而且要动态地变化。例如，飞行模拟器显示的视图就应当是从一架飞行飞机的驾驶舱所看到的。为了产生运动的效果，该系统每秒钟应多次产

生并显示一个新的稍微不同的视图。模拟器已用于训练宇宙飞船、飞机和轮船的驾驶员，最近，还用于训练汽车驾驶员。

汽车、飞机和房屋等三维物体的设计者想要看一看他们初步设计的样子。与构造模型或原型相比，产生真实的由计算机生成的图像往往是一个比较容易的、代价较小的和更为有效的看
424 初步结果的方法，并且还允许考虑更多不同的设计。若设计工作本身也是用计算机辅助进行的，那么，目标对象的一种数字化的描述也已经可用于产生图像。理想情况下，设计者还可依据所显示的图像交互地修改设计。已经开发了一个由汽车设计系统，用来决定在各种光线条件下，一辆汽车看起来是什么样子。真实感图像还往往与程序相结合，用来分析正在设计中的对象的其他方面，例如，其质量特性或对应力的反应等。

计算机成像术广泛地应用于娱乐界，不论是传统的动画卡通，还是作为商标徽记、广告宣传和科学幻想电影的真实感和超现实的图像。计算机产生的卡通可仿制传统的动画，还可以借助于引入更复杂的动作和更丰富或更有真实感的成像术而胜过手工技术。某些复杂的真实感图像的生产成本低于从该对象的物理模型所拍摄的照片。现在已生成出另外一些由真实模型实施极为困难或者不可能的图像。为娱乐表演创建的专用硬件和软件包括复杂的绘画系统、产生特殊效果和组合图像的实时系统。随着技术的进步，家庭和游乐场所的视频游戏产生出真实感不断提高的图像。

真实感图像正变成研究和教育方面的一种基本工具。一个特别重要的实例是在分子建模方面图形学的应用，如彩图22所示。有趣的是，在这里真实感的概念是如何展现的：真实感的绘图并不是“真”的原子，而是模仿的球和棒的形态和体积的模型。这允许建造比物理模型可能达到的更大的结构，并且它还允许有特殊效果，例如，表示反应逼真的震动键和色彩变化。在一个宏观的尺度上，在JPL制作的电影显示出NASA的空间探索任务，显示在彩图23中。

真实感成像术的另一个应用是在指挥和控制方面，其中，用户需要有关的信息并控制由图片表达的复杂过程。与模拟不同，模拟是试图去模仿用户将有效看到和感觉到的情形，而命令和控制的应用往往是针对所做的决策产生符号显示，强调某些数据并抑制另一些数据。

12.2 基本的困难

完全实现视觉真实感的基本困难是真实世界的复杂性。请注意你周围环境的丰富多彩。有许多表面纹理、精细的色彩灰度、明暗阴影、反射和周围物体的细微的不规则性。请想一想起皱的衣服上的图案、皮肤的皱纹、蓬乱的头发、在地板上磨损的痕迹以及碎裂的墙上油画。所有这些组合起来产生了一个“真实的”可视的体验。为模拟这些效果，计算代价可以很高：在
425 高性能计算机上产生彩图，可能要花几分钟甚至几个小时。

在探索真实感方面，更容易满足的子目标是提供充分的信息让观察者理解几个对象之间的三维空间关系。这个子目标能以一个低得多的代价来完成，它也是在CAD和许多其他应用领域中一个通常的要求。虽然，高度真实感的图像传达了三维空间关系，但它们也往往传达了更多的信息。例如，图12-1是一个简单的线条图形，但却令人信服地告诉我们：一个建筑物的一部分是在另一个建筑物的后面。这并不需要显示铺砌瓦片和砖块的建筑物的表面，也不需要显示建筑物投射的阴影。实际上，在某些相关情况中，这些特殊细节也许只会分散观察者对正被描述的更重要的信息的注意力。

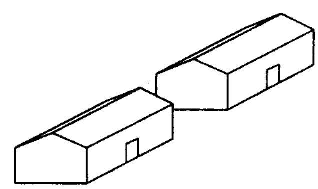


图12-1 两幢房屋的线条图

在描述空间关系方面，一个长期存在的困难是：绝大多数显示设备是二维的。因此，必须将三维物体投影成二维，投影会伴随着相当可观的信息丢失，有时这会在图像上产生模糊性。本章介绍的一些技术可用来将我们在可视环境中通常存在的这类信息找回来。使得人们的深度感知机制能够妥善地解决遗留的歧义性问题。

看一看图12-2a中的Necker立方体幻觉的例子，即一个立方体的二维投影。我们确实不知道它所表达的是图12-2b中的立方体还是图12-2c中的立方体。观察者很容易在二者间捉摸不定。这是因为图12-2a没有包含一个无歧义性解释的足够信息。

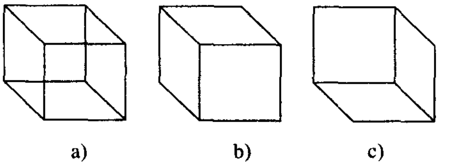
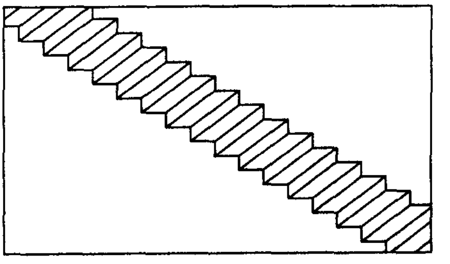


图12-2 Necker立方体幻觉。a)中的立方体相应于b)中的立方体还是相应于c)中的立方体呢

观察者关于显示对象的情况知道得越多，他们就越易于构造Gregory所谓的*目标假设*[GREG70]。图12-3显示出Schröder 阶梯的例子——我们认为它是向下看的阶梯呢？还是由下边看的向上的阶梯呢？我们喜欢选择前一种解释，也许是因为我们更经常地看到阶梯在我们的脚下，而不是在我们的头上，并且因此对从上面观察的阶梯“知道”得更多。然而，只要对想像稍一延伸，我们就能观察到这个图形的另一种解释。可是，只要眼睛一眨，对许多观察者来说就会发生相反的情况，阶梯又再一次表现为从上面观察到的。当然，加上相关的条件（例如一个人站在一级台阶上）将解决这一歧义性问题。


图12-3 Schröder阶梯的例子。这是由上面观察的阶梯还是由下面观察的阶梯

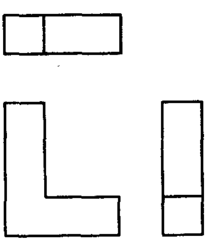
在随后的几节，我们将沿着通向真实感图像的路径列出一些步骤。这条路径实际上已是一组纠缠在一起的小路，而不是单一的笔直的道路。但为了简单起见，我们已经将它线性化了，提供了一个简捷的描述。我们陈述的第一种技术可应用到静态线条图。这些方法集中到在几个物体的二维显示中表示出三维空间关系的途径。接着介绍明暗图像技术，它靠光栅图形硬件来实现，它集中于光线与物体的相互作用。其后讨论逐步增加模型复杂度和动态性的问题，这适用于线条和有明暗的图片。最后，我们讨论真三维图像的可能性、显示硬件的进展，以及在完全交互的环境合成中图片生成的未来地位问题。

12.3 线条图的绘制技术

本节中，我们集中讨论真实感问题的一个子目标：在二维平面上显示三维的深度关系的问题。第6章所定义的平面几何投影可服务于此目标。

12.3.1 多正交视图

最容易产生的投影是平行正交投影。例如，平视图和立视图，其中的投影平面是垂直于一个主轴的。由于抛弃了深度信息，平视图和立视图通常是一起显示的，如图12-4中一个“L”型字母块的顶视图、前视图和侧视图。这种具体画法不难。然而，从一组这样的视图理解复杂的机械零件图也许需要研究好几个小时。当然，训练和经验会提高人的解释能力，对所表达的物体类型的精通可促进一个正确的对象假设的形成。但是，像彩图


图12-4 “L”型字母块的顶视、前视和侧视正交投影

24中所示的“标准场景”那样复杂的场景，在只显示三个这样的投影时，往往也是令人困惑的。426~427 虽然，单个点可以无歧义性地从三个互相垂直的正交投影来定位，但做这种投影时，多个点和线也许会一个掩盖另一个。

12.3.2 透视投影

在透视投影中，一个物体的大小与它和观察者的距离成反比。图12-5中所示的一个立方体的透视投影反映了这种比例。然而，仍然有歧义性问题。这个投影好像是一个像框，或者是被削除顶部的金字塔的平行投影，或者是两个面相等的矩形六面体的透视投影。如果假想的物体是一个被除顶部的金字塔，那么，较小的方形表示更接近于观察者的面；如果假想的物体是一个立方体或矩形六面体，那么，较小的方形表示远离观察者的面。

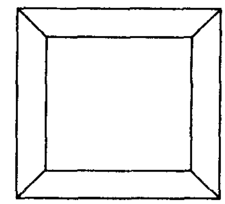


图14-5 一个立方体的透视投影

我们对透视投影的解释往往是基于这样的假定，即较小的物体在较远的地方。在图12-6中，我们或许假设较大的房子更接近于观察者。然而，至少当没有其他线索暗示时，例如，没有树或窗户时，看起来较大的房子（或许是一个大楼）实际上可能比一个看起来较小的房子（如一个小别墅）的距离更远。当观察者知道被投影的物体有许多平行线时，透视投影会进一步帮助传达深度信息，因为平行线会汇聚到它们的灭点。与尺寸减小的效果相比较，这种汇聚实际上是一种更强的深度信息。彩图25显示了一个标准场景的透视投影。

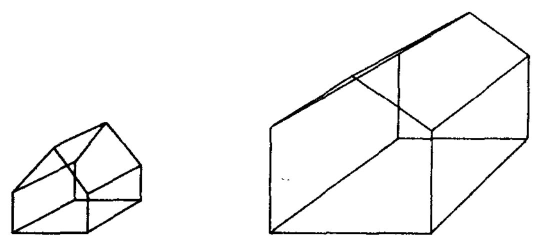


图14-6 两幢房屋的透视投影

12.3.3 深度提示

一个目标的深度（距离）可以由图像的亮度来表示：有意使距离观察者较远的那部分物体以较低的亮度显示。这种效果就是所谓的**深度提示**。深度提示利用了这样的事实：远的目标比近的目标显得更加朦胧，尤其是穿过薄雾观察看时更是如此。作为**空中透视**，画家用亮度变化（以及纹理、清晰度和颜色）描绘距离，这样的效果是令人信服的。因此，深度提示也可以看成是大气散射效果的一个简化形式。

在向量显示中，深度提示是沿一条向量进行光线亮度的插值来实现的，插值是以该向量起
428 点和终点z坐标的函数来进行的。彩色图形系统经常推广这一技术来支持在图元的色彩与用户指定的深度提示色彩之间进行插值，后者一般是背景色。为了将效果限制在有限的深度范围内，PHIGS PLUS允许用户指定前、后两个深度提示平面，在它们中间存在深度提示。按每个平面分别赋给比例因子，表征用于前平面之前和后平面之后的原始颜色的比例和深度提示颜色：两平面之间点的颜色在这两个数值之间进行线性插值。肉眼的亮度分辨率低于空间分辨率，所以，对于精确地描绘很小情况的距离差别，深度提示是没有用的。当然，在描绘大的距离差别时，它十分有效，或者在描述小的距离时，它可作为一种放大的提示信息。

12.3.4 深度裁剪

深度裁剪可提供更进一步的深度信息。安放了一个后裁剪面，切过正在显示的物体。观察者知道用裁剪平面裁掉了部分物体。也可使用前裁剪面。借助于允许一个或两个平面的位置动态变化，系统可对观察者传达更多的深度信息。后平面深度裁剪可被想像成深度提示的特殊情

况：在普通的深度提示中，亮度是z坐标的平滑函数；在深度裁剪中，它是一个阶跃函数。一种与深度裁剪有关的技术是借助于在物体上与某些面相交的所有点形成高亮显示。当裁剪平面动态地移过物体时，这种技术特别有效，并且已经用于协助说明沿第四维的深度[BANC77]。

12.3.5 纹理

简单的向量纹理（例如，**断面线**）可用于一个物体。这些纹理伴随着一个物体的形状并描绘出更清晰的轮廓线。对完全相同的一组面之一加上纹理可澄清一个潜在的歧义性投影。在透视投影中，加纹理特别有用，因为它加上了许多线，其汇聚和透视缩小特性可提供有用的深度提示。

12.3.6 颜色

颜色可用于象征性地将一个物体与另一个物体区别开，方法是每个物体都赋予一种不同的颜色。颜色也可用于线条图，以提供其他信息。例如，一个物体中每个向量的颜色也可由颜色的插值来确定，它是向量端点的温度的编码。

12.3.7 可见线的判定

我们指示的线条图的最后一个技术是**可见线判定**或**隐藏线消除**，其结果是在屏幕上只显示 429
可见的（也就是不被遮挡的）线或线的一部分。惟有以边（线）为界的面可能挡住其他的线，这样，挡住其他物体的那些物体必须是面的集合或者是实体。

彩图26显示出隐藏线消除的效果（也使用12.3.6节中描述的颜色）。由于移走了隐藏线，视图隐蔽了所有的不透明物体的内部结构，但它们并非显示深度关系的最有效方法。清除了隐藏线的视图提供的深度信息少于剖面视图，隐藏线以虚线显示可能是一种有用的折衷办法。

12.4 明暗图像的绘制技术

12.3节提出的技术可以在向量显示器和光栅显示器中建立线条图。本节所介绍的技术利用光栅设备的性能去显示有明暗效果的区域。当图片是用光栅显示器绘制时，问题是由于光滑的边沿和明暗效果必须在太粗糙的像素栅格上重新产生而引起的。最简单的绘制明暗图像的方法成为走样问题的牺牲品，这一问题最初在3.14节已遇到过。由于反走样技术在产生高质量图片中起着基础作用，所以本章中所有的图片都是用反走样技术来建立的。

12.4.1 可见面的判定

与可见线判定相类似，**可见面判定**或**隐藏面消除**则仅仅显示观察者可见的那部分面。正如我们曾见过的那样，没有可见线判定也可理解简单的线条图。当有少量的线时，在前面的那些线也许并不是观察它们后面的那些线的主要障碍。在光栅图形中，若表面绘制成不透明的区域，那么，对于制作有意义的图片而言，可见面的判定是至关重要的。彩图27显示一个例子，在其中，一个物体的所有面都画成同样颜色。

12.4.2 光照和明暗处理

彩图27的一个问题是每个物体都被表示成一个平板剪影。我们走向真实感的下一步是可见面的明暗效果。最终，每个面的外表应取决于照射它的光源的类型、它的特性（色彩、纹理、反射）以及它与光源、观察者和其他面的相对位置和方向。

在许多真实的可视环境中，大量的环境光从所有的方向照射过来。环境光是最容易模拟的一种光源类型，原因是，在一个简单光源模型中，假设它在所有的面上都产生恒定的光照，不 430
管表面的位置和方向怎样。然而，依靠环境光自身会产生非常不真实的图像，这是因为很少有真实环境是惟一的用均匀的环境光来照射的。彩图27是这种方法产生明暗效果的一个例子。

点光源（它的光线是从单一的点发出的）可近似于一个小的白炽灯泡。一个定向光源（它们

的光线全来自同一个方向）可用于表示一个远距离的太阳，可将它近似为一无限远的点光源。这些光源的效果依赖于面的方向，所以，模仿它需要增加额外的工作。如果面正交（垂直）于入射光线，它将被照射得很明亮；朝着光线的面越倾斜，照射得越少。当然，光照的变化对一个物体的三维构造是强有力的提示。最后，*分布的或扩展的*光源（它们的表面区域发出光线，例如，日光灯）要模仿它们更加复杂，这是因为它们的光线既非来自单一方向，也非来自单一的点。彩图28显示出以环境光和点光源对场景照射，并对每个多边形分别进行明暗处理的效果。

12.4.3 插值明暗处理

插值明暗处理是计算明暗信息的一种技术，它计算每个多边形顶点的明暗信息，在多边形的内部则通过插值来决定每个像素上的明暗。当意欲用一个多边形物体来近似一个曲面时，这方法尤其有效。在这种情况下，每个顶点上明暗信息的计算可基于在该点的面的实际方向，并可用于共享这个顶点的所有多边形。在一个多边形内对这些数值进行插值近似于在一个曲面内明暗程度的平滑变化，而不是在一个平面内变化。

即使物体假设为多面体而不是曲面，也可得益于插值明暗技术，其原因在于，对一个多边形的每个顶点计算出的物体明暗信息可以是不同的，但一般说来比曲面物体顶点的差别要小得多。当对一个真实多面体物体计算明暗信息时，对多边形顶点确定的值仅用于该多边形，并不用于共享该顶点的其他多边形。彩图29显示出Gouraud明暗处理技术，这类插值明暗技术在14.2节讨论。

12.4.4 材质属性

当确定物体的明暗程度时，若把每个物体的**材质属性**考虑进去将进一步加强真实感。有些材料是较暗淡的，并在各个方向均匀地减弱和分散反射光线，像一支粉笔那样。另一些是有光泽的，并且只在相对于观察者和光源的一定方向上反射光线，比如一面镜子。彩图31显示出当某些物体模仿成有光泽的时候，我们所看到的场景。使用Phong明暗处理模型，这是一种更精
431 确地用插值生成明暗图像的方法（14.2节）。

12.4.5 曲面造型

虽然插值明暗技术大大改善了图像的外观，但物体在几何上仍是多边形。彩图32用了包含曲面的物体模型，在图像的每一像素点上计算全部明暗信息。

12.4.6 改进光照和明暗效果

对大多数计算机所生成图像有“不真实”外观的最重要原因之一是，未能精确模拟光线与物体互相作用的许多方面。彩图33采用了更好的光照模型。14.1.7节讨论了有效的物理上正确的光照模型设计的进展。

12.4.7 纹理

正如12.3.5节所讨论的，物体的纹理不仅提供了附加的深度提示，而且，还可模拟真实图像的表面细节。彩图35表示出模拟表面纹理的多种方法，从表面色彩的变化（如以图案装饰的球）到表面几何形状的变形（如有条纹的环面和扭曲的圆锥）。

12.4.8 阴影

将一个物体的阴影投射到另一个物体，可导致进一步的真实感。请注意，这是在表现一个物体的可视面受另一物体影响时首先会遇到的。彩图35显示出由位于场景后方的灯构成的阴影。阴影增强了真实感并提供了附加的深度信息：如果物体A在B面上投射出阴影，那么，我们知道A是在B和一个直接的或反射的光源之间。一个点光源将投射出清晰的阴影，因为从任何一个点，不论它是完全可见的还是不可见的都很明确。一个扩展的光源投射出“柔和”阴影，因为从看到全部光源的那些点起，只见到一部分光源的那些点，直到完全看不见的那些点，有一个平滑地过渡。

12.4.9 透明性和反射

迄今为此，我们只讨论了不透明的面。在图片制作方面，透明的面也是很有用的。透明性的简单模型不包括光线穿过一个透明实体时的折射（弯曲）。然而，若透明度并非用来像显露出物体内部几何形状那样模仿真实性，那么，没有折射可以是一个明显的优点。更复杂的造型包括折射、漫射透明性以及随距离加大的光线衰减。类似地，光线反射模型可模仿一个理想的反射另一个物体的镜面反射模型或者模仿不太光泽的平面的漫反射。彩图36显示出地面和茶壶的反射效果；彩图41显示透明性。 432

像构造阴影一样，构造透明性和反射除了需生成明暗效果的表面以外，还需要其他表面的知识。更进一步，折射透明度是我们曾经提到的第一个效果，它要求实际造型的物体应作为实体而不单是作为表面。我们必须知道有关光线所穿过的材料的某些情况以及正确地模仿折射时，光线所经过的距离。

12.4.10 改进的相机模型

至此，所显示的图片都是基于针孔透镜和无限快的快门这样一种相机模型的。所有物体都是以精确的聚焦并在一瞬间反映出来的。更精确地模仿我们（和相机）看世界的方法是有可能的。例如，借助于透镜焦点特性的模拟，我们可产生彩图37那样的图片，它们显示出**景深**：物体的某些部分在焦点中，那些更近或更远的就在焦点之外。其他一些技术允许特殊效果的应用，例如，鱼眼透镜。许多早期用计算机产生的超现实图片部分地反映出缺乏景深的效果。

以通常的静态或移动相机所取得的照片上的移动物体看起来不同于静止物体。这是由于快门在一个有限的时间内打开，移动物体的可见部分在胶片底板上是模糊的。这种效果称为**运动模糊**，在[KORE83]中被模仿。运动模糊不仅在静止中取得运动效果，而且在高质量的动画中也是极为重要的，如[FOLE90]的第21章所述。

12.5 改进的物体模型

与所用的绘制技术无关，探索真实性的工作也部分集中于建立更可信的模型的方法上，无论是静态的还是动态的。有些研究者已开发出特定类型物体的模型。例如，雾、波浪、山和树；见彩图11、彩图12、彩图13。基于分形、基于文法方法及粒子系统可生成这些图像。其他研究者还集中精力在样条、过程模型、体绘制、基于物理建模及人体建模方面。另一重要的题目是大量物体的自动定位问题，如用对森林中树木进行定位太枯燥了以致无法用手工来完成。Witkin和Kass[WITK88]描述了一个应用于模型Luxo Jr.动画的自动放置方法。然而，出现在本书封面的 Luxo Jr. 图像来自于Pixar公司的动画[PIXA86]，它们并不是采用自动放置方法来生成的。9.5节讨论了其中某些技术，而更详细的内容可在[FOLE90]的第20章中找到。 433

12.6 动力学与动画

12.6.1 运动的价值

所谓**动力学**，指的是在一系列图片中充满着变化，包括位置、大小、材质属性、光照和观察规范的变化等，实际上指任何部分的场景或应用于它的技术的变化。在走向更为真实的静态图像的过程中，动力学的好处是可以独立加以考查的。

最普遍的动力学类型是运动动力学，从简单的在用户控制下完成的变换到复杂的动画。从计算机图形学领域出现以来，运动已成为一个重要的部分。在早期的低速光栅扫描图形硬件的

时代，运动能力是向量图形系统具有很强竞争力的一个卖点。如果将同一物体的一系列投影（每个都是从同一物体周围稍微不同的视点取得的）快速、连续地显示出来，那么，该物体看起来是在旋转。借助于这些视图的整体信息，观察者可建立一个物体假设。

举个例子，一个旋转的立方体的透视投影可提供几种类型的信息。有一系列不同的投影，它们本身是有用的。在这种情况下，靠近旋转中心的点与远离旋转中心的点相比，其最大线速度较低，这就是靠运动效果来补充的。这种区别可帮助弄清楚一个点离开旋转中心的相对距离。此外，在透视投影下改变距离时，立方体不同部分的大小变化提供了有关深度关系的附加提示。当运动在观察者交互控制下时，它会变得更有用。借助于有选择的变换一个物体，观察者也许有可能更快地构成一个物体的设想。

与用简单的变换去弄清复杂模型相反，若在真实感的形态中运动，令人惊讶地简单造型看起来极有说服力。例如，位于一个人体模型的关键部分的少数几个点自然地运动时，可提供运动中人体的令人信服的例证。这些点本身看起来并不像一个人，但它们确实告诉观察者一个人是存在的。人们很清楚地知道，绘制运动中的物体比表示静态物体需要更少的细节，这是因为观察者更难于拾取出正在运动中的物体的细节。例如，电视观众往往很惊讶地发现单独的一帧电视画面看起来是多么贫乏，并带有颗粒状。

12.6.2 动画

动画，顾名思义，就是使静止的画面活动起来。人们通常认为动画与运动同义，但实际上
434 动画包括一切具有视觉效果的变化。也就是说，它包括物体的位置（**运动动力学**）、形状、颜色、透明度、结构和纹理（**更新动力学**）等随时间的变化，以及光照、相机的位置、方向或焦距乃至绘制方法的变化。

动画在娱乐业得到了广泛的运用，也被使用于教育、工业应用（诸如控制系统、智能终端和飞行模拟器）以及科学研究。计算机图形（特别是动画）在科学计算这方面的应用已逐渐归结到**科学计算可视化**这一主题之下。不过，可视化不仅仅是将图形学应用于自然科学和工程学中，它还涉及其他学科，譬如信号处理、计算几何和数据库理论。通常，科学计算可视化中的动画是对自然现象的模拟。这些模拟的结果可能是表示二维或三维数据的大型数据集（例如，对流体的模拟）；这些数据首先被转换为图像，而后组成动画。另一方面，模拟的结果也可能生成实际物体的位置和运动方向，而后，它们必须以某种特定方式进行绘制来产生动画。比如说，对化学过程的模拟就属于这种情形，模拟会生成各式各样的、相互作用的原子的位置和运动方向，但动画所显示的却可能是表示分子的一个个球-棍结构，或是表示原子的相互遮盖的光滑的球体。某些情况下，模拟程序会包含一种嵌入式的动画脚本语言，这时，模拟和动画的过程是同时进行的。

如果因每秒播放的帧数过多而造成动画中视角变化太快，就会发生**时域走样**。这方面的例子有：车轮有时看起来会向后转，物体在刹那间穿过很大视角范围时所产生的跳跃运动。录像的播放速率是每秒30帧，电影通常是每秒24帧，两者都能为许多应用提供合适的效果。当然，要利用这样的速率，就必须为录像或电影的每一帧创建一个新图像。如果动画制作者不这么做，而仅为每两帧录像记录一幅图像，则所得到的动画有效帧速率仅为每秒15帧，并且看起来有跳动。

传统的动画（也就是非计算机动画）本身就是一门学科，我们不准备全面探讨它。在这里，我们着重介绍计算机动画的基本概念和一些经典的系统。我们将首先探讨传统动画及现有的计算机辅助动画设计的方法。然后，我们的重点将转向计算机动画。因为这些动画中大部分是三维的，所以传统的二维角色动画所采用的许多技术不能被直接沿用。而且，当动画制作者不是直接用手来绘制动画时，对动画流程的控制也更加困难：描述某件事该怎么做比直接动手做这件事要难得多。因此，在讲解了各种计算机动画语言后，我们将深入学习几种动画控制技术。

在本章末尾，我们再讲解动画的一些基本规则和动画所特有的一些问题。

1. 传统动画和计算机辅助动画

传统动画的制作流程是很固定的：第一步是写动画剧本（也可能仅打打腹稿），下一步是建立**故事板**。所谓故事板，是一种大纲形式的动画——能体现动画结构和构思的、高度简化的草图序列。第三步，录制声道（假如有的话）；接着，制作详细布局（为动画中每个场景画一张图），同时读入声道——这意味着，各种声音是以其出现时刻为序依次录制的。这样，上述的详细布局和声道之间就相互关联了。再下一步，画出动画中特定的**关键帧**——在这些帧中，动画中的运动物体处于指定位置，由此可以推算出物体在中间各帧中的位置。然后，在关键帧之间插入这些中间帧（这个过程被称为**插值**——inbetweening），这样，初步的胶片就完成了（上述整个过程被称为**素描**（pencil test））。而后，用彩笔手工复制或直接拍照，将素描的各帧转换为**醋酸盐胶片**。在多层动画中，使用了多层醋酸盐胶片，一部分作为背景的胶片保持不变（但也可能要做一定的平移），而另一些前景对象则随时间而变化。对胶片进行着色后，将它们按正确的次序进行组合，就可以正式录制了。制作动画的人员有着十分明确的分工：一些人设计剧本，一些人画关键帧，还有些人完全是插值帧制作员，其余的人仅仅是为最终的胶片着色。因为关键帧和插值帧的使用，这种动画被称为**关键帧动画**。这个名字也适用于模拟上述过程的基于计算机的系统。 435

传统动画的许多阶段用计算机实现似乎很理想，特别是插值阶段和着色阶段，它们可使用种子填充技术[SMIT79]在计算机上加以实现。但是，在用计算机进行处理之前，图像必须被数字化，其途径有：使用光学扫描，在数据输入板上对图像进行跟踪，或一开始就用绘图软件来生成原始图像。所得到的数字图像也许还得进行后期处理（例如滤波）来消除输入过程（特别是光学扫描）所产生的干扰，并对图像的轮廓进行适当的平滑。

2. 计算机动画语言

已经开发了许多用于描述动画的计算机语言。它们的范围从单机专用符号语言到与通用语言一起使用的过程包（见[FOLE90]的第21章）。许多计算机动画语言是和建模语言混合在一起的，因此，对动画中的物体的描述和这些物体的动画是同时完成的。

3. 动画控制方法

动画控制在某种程度上与描述动画所用的语言无关——绝大多数控制机制可用不同类型的语言实现。动画控制机制涉及的范围从完全显式的控制技术到由基于知识系统所提供的高度自动化的控制技术，前者由动画制作者使用平移、旋转及其他对物体的位置或属性进行改变的操作来显式地描述场景中每个对象的位置和属性；后者对动画进行高度抽象的描述（如“使人走出房间”）并产生所要的显式控制，这种控制对完成动画制作中的变化是必要的。一些控制动画的更新的技术在[FOLE90]中描述。

4. 动画的基本规则

从1925年到20世纪30年代末，传统的角色动画由一种艺术形式发展为类似沃特迪斯尼工作室这样的工业。起初，动画所需要的不过是一系列的卡通图片——静止图像的一个集合，将它们组合在一起，就成了动画。在动画技术不断发展的同时，其中所涉及的一些基本原则成为了角色动画的基本规则，而且在今天仍被使用[LAYB79；LASS87]。尽管这种原则来源于卡通人物动画，但许多同样适用于逼真的三维动画。这些规则以及在三维人物动画中它们的运用在[LASS87]中有全面的评述。在这里，我们仅介绍一些最重要的规则。我们知道这些规则并不是绝对的，这一点十分重要。就像许多现代艺术已脱离绘画的传统框架一样，现代的动画制作者也已脱离了传统动画的框架，而且其作品通常十分精彩（例见[LEAF74；LEAF77]）。在这 436

些规则中，挤压和拉伸通过变形来指示物体的物理性质；渐入和渐出运动提供了平滑的变化；正确的分阶段，或通过投影有关事件的最重要信息来选择视图，以替换动画。

5. 动画特有的一些问题

正如从二维图形到三维图形的过渡产生了许多新问题和新的挑战一样，从三维过渡到四维（加入时间维度）也有其特有的问题。其中之一就是*时域走样*。就像二维图形和三维图形的走样问题可由提高屏幕分辨率部分地加以解决一样，动画中的时域走样问题可由提高时域分辨率部分地解决。当然，二维图形对此的另一解决办法是反走样；三维图形中相应的办法是时域反走样。

12.7 体视学

迄今为止，我们曾讨论过的全部技术都是对观察者的双眼呈现相同的图像的。现在做一个实验。先用一只眼看桌面，然后，再用另一只眼看。由于我们的一双眼睛相互分开几英寸，两次观察稍有区别，如图12-7所示。由于这种分离产生的*双眼视差*提供了一个称为**体视学**（**立体观测**）或**立体视觉**的强有力的深度提示信息。我们的大脑融合这两个分离的图像为一个图像，并被解释成三维的。这两个图像被称为*立体对*。一个世纪以来，立体对广泛地用于立体观察者，今天已用于普通玩具观察大师（View-Master）。彩图22显示一个分子的立体对。你可以在两个图像之间放置一个固定的纸片并垂直于图像面，使你可借助于一只眼看一个图像，从而可以融合两个图像成为一个三维图像。有些人不需要这个纸片也可看到这种效果，而少数人却完全看不到。

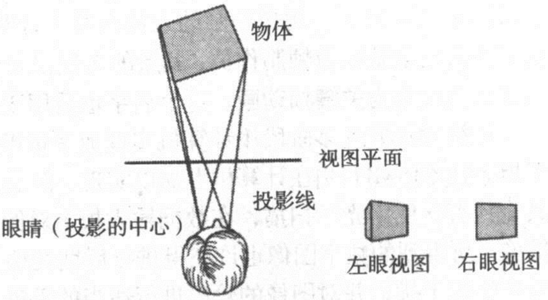


图12-7 双眼视差

存在着对每只眼提供不同图像的其他一些技术，包括具有偏振光滤波的眼镜和全息术。这些技术中，有些可使得占有空间的真三维图像成为可能，而不是被投射在单一的平面上。这些显示可以提供一个附加的三维深度感：正如在真实生活中那样，靠近的目标实际上是更靠近，因而，根据每物体的接近情况，观察者的眼睛可有区别地聚焦在不同物体上。在习题6.17中描述了立体投影的数学基础。

12.8 改进的显示技术

除了改进用于设计和绘制物体的软件之外，显示技术本身的改善已提高了真实感的幻影。计算机图形学的历史，部分是显示设备所达到的视觉质量稳定提高的历史。还有，现代监视器的色域和动态灰度范围是我们所能看到的两个小子集。在显示器的图像要达到印刷精美的职业摄影照片的清晰度和对比度方面，我们还有一段很长的路要走。有限的显示分辨率使它不可能复制出极为精美的细节。可见的磷光图像、来自屏幕的刺眼、几何畸变以及帧频闪烁的闪光效果等都是我们正观看着的显示器的各种各样的问题。与我们的视野相比，显示器相对较小的尺寸也提醒我们，显示器只是一个世界的窗口，而不是世界本身。

12.9 与其他感官的交互

走向真实感的最后一步，也许是将真实感图像与呈现给我们的其他感官的信息集成在一起。

计算机图形学有一段很长的编程历史，它依赖于多样化的输入设备允许用户交互。飞行模拟器是图形技术与真实的发动机声音和运动结合的当前实例，所有这些由一个驾驶员座舱的大模型提供，并建立起一个完整的环境。头带模拟器监视着头部的运动，使得另一个称为*头运动视差*的重要三维深度提示成为可能：当用户的头部从一边移到另一边时，也许试图去看更多的部分被隐藏的物体，那么视图的变化就像它在真实的生活中应有的那样。头盔显示器的另一个活跃的工作在于对*虚拟世界*的探索。例如，还没有构造好的分子或建筑物的内部 [CHUN89]。 437~438

许多现代游乐场的特色是让游戏者乘坐的小汽车或飞机随时间运动，并模拟综合的和数字化的图像、声音和力反馈等。这种附加的输出和输入通道的使用指出了通向未来的系统的道路，它将提供完全沉浸在包括听觉、触觉、味觉和嗅觉在内的各种感受。

小结

本章中，我们提供了有关真实感图像技术的一个高层次的介绍。在下面几章中，我们将详细讨论这些技术是如何实现的。当你读后面几章出现的算法时，你的脑子中应当记住五个关键问题：

1) *这个算法是一般的算法还是具有特殊目的的算法？*某些技术仅在特定环境中是好的，而另一些则设计得更为通用。例如，某些算法假定所有的物体是凸的多面体，并从这个假定出发，导出它的速度和相对简单的部分。

2) *算法的空间–时间特性是什么？*数据库的大小和复杂性等这样的因素如何影响算法？或者图片绘制的分辨率又如何影响算法？

3) *所产生效果的可信度如何？*例如，折射光照模型是正确的吗？它看起来只是在某些特定情况是正确的，抑或完全不能模拟？可否增加效果？例如，阴影和镜面反射是否可加上去。如何能确信它们？牺牲据以表现效果的精度也许使得有可能在程序的空间或时间要求方面有重要改善。

4) *给定一个生成图片的目的，该算法是否适当？*在随后各章中，许多图片背后的基本原理可以归结为“如果它看起来是好的，就这么做”。这个指导思想可从两方面来解释。如果一种简单或快速的算法能产生吸引人的效果，即使在物理法则上找不到正当的证明，那也可以使用它。另一方面，如果一种算法昂贵得可怕，而对于绘制某种效果，它是惟一已知的方法，你就可以使用它。 439

习题

12.1 假定你有一个图形系统，它可以实时地画出任何一种在本章中所引用的彩图，试考虑你所熟悉的几种应用领域。对每个领域，列出那些最有用的效果和最没有用的效果。

12.2 请说明从一个单色的、旋转的线框立方体的正投影图中不可能推论出它的旋转方向。请解释怎样附加技术就可以在不改变投影的前提下帮助弄清旋转方向。 440

第13章 可见面的判定

对于一组给定的三维物体与观察（视见）说明，我们常常需要确定物体上的哪些线或面是可见的，从而在绘制时能够只显示这些可见的部分。这种可见性在透视投影中是相对于投影中心的，在平行投影中则是相对于平行投影方向的。这一确定可见性的过程称为**可见线判定**或**可见面判定**，也可称为**隐藏线消除**或**隐藏面消除**。在可见线的判定过程中，线可以看成是不透明面片的边，这些面片对远离视点的面片的边产生遮挡。因此，我们可以将整个过程通称为**可见面判定**。

尽管这一基本思想的表述非常简单，实现起来却要求计算机具有很强的处理能力，并且在常规的机器上需要花费大量的运算时间。为解决这个问题，人们提出了许多构思精巧的可见面判定算法，这将在本章中进行介绍。另外，人们为此也设计了一些专用的体系结构，其中一些将在[FOLE90]的第18章中讨论。可见面的判定有两种最基本的解决方案，在两种方案中都将每一对象看成是由一个或多个多边形（或更复杂的面片）组成的。

第一种方案是在生成图像中的每一点时确定n个对象中的哪个点可见。该方案可用伪代码表示如下：

```
for （图像中的每个像素） {
    确定由投影点与像素连线穿过的距
      观察点最近的对象；                                   441
    用适当的颜色绘制该像素；
}
```

最直接的一种方法是对每一像素考察全部n个对象，判定沿投影点穿过该像素的对象中哪一个离视点最近。对p个像素而言，其计算量正比于np，其中p在高分辨率显示中会超过10^6。

第二种方案是直接在对象之间互相比较，以除去完全不可见的对象或不可见的部分。下面的伪代码便体现了这种思想：

```
for（世界坐标中的每一个对象）{
    判定对象中未被遮挡的部分，包括其自身的遮挡
        与其他对象的遮挡；
    用适当的颜色绘出可见部分；
}
```

我们可以简单地通过将对象与其自身以及与其他对象的比较来除去不可见的部分。这样做的运算量正比于n^2。表面上看，由于$n<p$，似乎第二种方案更为高效，但实际上，它的每一个基本的步骤都很复杂且费时，因此，第二种方案往往更慢且更难实现。

人们常常称这两种方案为**图像精度算法**与**对象精度算法**。图像精度算法是基于显示设备的分辨率来判定每一个像素的可见性的算法，而对象精度算法则是基于对象定义的精度来判定每一对象可见性的算法。[⊖]由于对象精度的遮挡计算与显示分辨率无关，所以最终少不了用特定的

⊖ 术语图像空间和对象空间（由Sutherland、Sproull和Schumaker[SUTH74a]推广）常用来表示相同的区分。但是，这些术语在计算机图形学中还常常代表不同的意义。例如，图像空间用来表示透视变换后[CATM75]或投影到视图平面后[GILO78]在其原有表示精度下的对象。为了避免混淆，我们稍加修改了我们所用的术语。我们对于对象的透视变换或投影将以明显的方式表达出来，并用术语图像精度和对象精度来指示完成计算所采用的精度。例如，如果原有对象定义的精度不变，两个对象不在一个投影平面上相交的操作则是对象精度的操作。

分辨率来显示这一步。如果生成图像的大小发生改变，只有最后显示这一步需要重新计算，比如在光栅显示器上显示相应的不同数目的像素。这是由于每个可见对象投影的几何图形是用对象数据库的精度表示的。而图像精度算法则与此形成对比。例如，用图像精度算法将一幅图像进行放大时，原来用低分辨率生成的可见面图像必须重新计算以表现更多的细节。这也就是图像精度算法在计算可见性时存在走样问题而对象精度算法不存在这个问题的原因。

对象精度算法最初是为向量图形系统开发出来的。在这些设备上，隐藏线的消除是非常自然的
442 事情：完全被其他面遮挡的线被去除；部分被遮挡的线被裁剪成一条或多条可见的线段。所有的处理都在原始的线条集合中完成，并生成相同格式的新的集合。相比较而言，图像精度算法则是为光栅显示设备编写的，其目的是减少需要进行可见性判定的点的数量。这是一种很自然的选择。向量显示具有较大的地址空间（在早期的系统中就有4096×4096），但能够显示的线条与对象的数目有很大的限制；而光栅显示设备就不同，它的地址空间非常有限（早期系统为256×256），却具有显示无限多个对象的潜力。后续的算法常常结合了对象精度算法和图像精度算法这两种算法的思想，用对象精度算法来获得精确度，而用图像精度算法来获取绘制速度。

本章的内容是这样安排的：首先我们介绍一些与一般可见面算法的效率相关的问题，然后，我们给出判定可见面的主要方法。

13.1 有效可见面判定算法的技术

典型图像精度算法与对象精度算法的公式可能需要许多运算量较大的操作。这些操作包括确定对象和投影线，求两个对象的投影，判断两个投影是否相交以及交于何处。对每一次相交运算，都必须计算离观察者最近的可见对象。为了尽可能减少创建图片所需的时间，我们希望使这些费时的操作以尽可能少的时间与尽可能少的次数执行。以下各节便给出了几种相关的技术。

13.1.1 相关性

Sutherland、Sproull与Schumacker[SUTH74a]提出可见面判定算法能够利用**相关性**——环境或它的投影所表现出来的局部相似程度。在我们所需要绘制的环境中常常存在这样的对象，它们的属性从一部分到另一部分发生着平滑的变化；或者说，在一幅图像中我们怎样区分不同的对象，依赖于对象的属性值（包括深度、颜色及纹理等），这些属性很少有不连续点。这种相关性使我们在完成环境或图像的某一部分的计算时，对其相邻部分的计算可以重用这一部分的计算结果，包括完全照搬的重用和在其上进行适量增减的变化。与重新计算每一部分相比，这样做大大提高了运算效率。这种相关性可以概括为以下几类[SUTH74a]：

443
- **对象相关性**。如果一个对象与另一个对象完全分离，那么就只需要对这两个对象之间进行比较，而不必对组成对象的面与边进行比较。例如，对象A的每一部分都比对象B远离视点，那么就不需要判断A的面是否遮挡B的面。
- **面相关性**。指一个面上的曲面属性发生连续的变化。这样，对某一部分的计算只要经过适量的修改便同样适用于其相邻部分。在有些模型中，面与面之间能够保证不互相穿插。
- **边相关性**。指一条边只有在从一条可见边后面穿过或穿过一可见面时才会改变其可见性。
- **隐含边相关性**。两平面相交时，它们的交线（隐含边）可由相交的两点确定。
- **扫描线相关性**。指图像中一条扫描线上的可见对象跨段与其相邻的扫描线上的可见对象跨段相差甚小。
- **区域相关性**。指一组相邻的像素被同一可见面所覆盖。一种特例是*跨段相关性*，即面在扫描线上一组相邻像素的跨段上的可见性。

- **深度相关性**。指同一面的相邻部分深度值相近，而不同面在屏幕同一位置的深度值相距甚远。在某一面上，一旦确定了一点的深度值，其余点的深度值常常可以通过一个简单的差分方程求出。
- **帧相关性**。同一环境在两个相邻的时刻生成的图像有可能非常相似，尽管其对象与视点发生了小的改变。因此，一幅图像的计算可以重用于下一幅图像中。

13.1.2 透视变换

可见面判定显然应该在投影到二维图像之前的三维空间完成，因为二维变换破坏了深度比较所需要的深度信息。撇开选择的投影方式不谈，在一个点处的基本的深度比较可以简化为这样的问题：给定点$P_1 = (x_1, y_1, z_1)$与$P_2 = (x_2, y_2, z_2)$，它们是否互相遮挡？也可以这样问，P_1与P_2是否在同一投影线上（见图13-1）？如果回答是肯定的，那么需要对z_1与z_2进行比较以判定哪一点离视点更近；反之，则两点之间不存在遮挡关系。

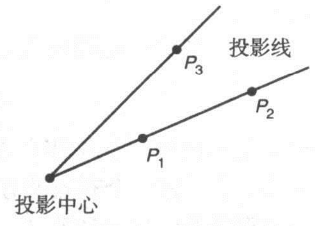


图13-1 P_1与P_2在同一条投影线上，则较近的一点遮挡较远的一点；否则，不形成遮挡关系（如P_1不遮挡P_3）

深度比较通常在规格化变换（见第6章）之后进行，这样，在平行投影中，投影线方向平行于z轴，而在透视投影中则从原点向外发散。对平行投影而言，如果两点的$x_1 = x_2$且$y_1 = y_2$，则两点在同一投影线上。而对透视投影而言，我们必须做四次除法，即判断$x_1/z_1 = x_2/z_2$且$y_1/z_1 = y_2/z_2$，才能确定两点在同一投影线上（见图13-1）。更进一步，如果P_1要与另外的一点P_3进行比较，还需要再多做两次除法。 444

为了免去这些不必要的除法运算，可以先将三维的对象变换到三维的屏幕坐标系。这样，变换后对象的平行投影就与未变换对象的透视投影一致了。从而使点与点之间的遮挡测试也与平行投影中相同。这种透视变换将对象进行变形，同时将投影中心移动到z轴的正无穷远处，使投影线互相平行。图13-2表示的是该变换的透视投影视见体的效果。图13-3是一个立方体的变形效果图。

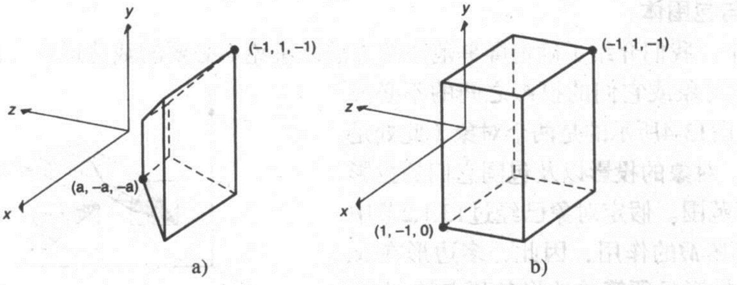


图13-2 规格化的透视投影视见体。a)透视变换前，b)透视变换后

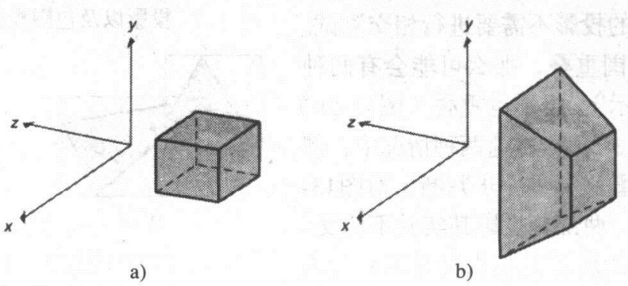


图13-3 立方体。a)透视变换前，b)透视变换后

这一变换的实质是保存了相对的深度、直线和平面，同时也执行了透视缩小效应。在第6章中已经讨论过，完成透视缩小效应的除法运算仅对每一点执行一次，而不必在两点之间进行比较时执行。矩阵可表示如下：

$$M=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix},\quad 0>z_{\min}>-1 \tag{13-1}$$

该矩阵将规格化的透视投影视见体变换为长方体，其边界为

$$-1\leqslant x\leqslant 1,\qquad -1\leqslant y\leqslant 1,\qquad -1\leqslant z\leqslant 0 \tag{13-2}$$

规格化后的棱台视见体在进行M矩阵变换之前可以进行一定的裁剪，但裁剪之后的结果必须被矩阵M乘。另一个有效的方法是将M矩阵合成到透视规格化变换矩阵N_{per}中（见第6章），这样就只需要执行一次矩阵乘法。然后在做除法之前对齐次坐标进行裁剪。假设相乘后的结果是（X，Y，Z，W），那么，当$W>0$时，裁剪的边界为

$$-W\leqslant X\leqslant W,\qquad -W\leqslant Y\leqslant W,\qquad -W\leqslant Z\leqslant 0 \tag{13-3}$$

上式由式(13-2)中用X/W，Y/W，Z/W分别替换x，y，z而得，表明式(13-2)中的x，y与z是除以W后所得的值。裁剪之后再除以W便可得(x_p，y_p，z_p)。要注意的是，矩阵M假定视见体位于z轴为负的半空间中。为了表述上的方便，我们的例子常常用减少正的z值来表示远离视点。在另外一些采用将右手坐标系转换到左手坐标系的系统中，则正好相反。

现在，我们可以从图13-1所显示的复杂性中解脱出来，继续进行可见面的判定。当然，一旦确定了平行投影，透视变换矩阵M就不需要了，因为平行投影规格化变换N_{par}能够使投影线平行于z轴。

13.1.3 范围与包围体

在第3章中，我们介绍了确定屏幕范围的方法以避免不必要的裁剪运算。这里，我们用同样的方法来消除对象或它们的投影之间的不必要的比较运算。图13-4所示的是两个对象（此处是三维多边形）、对象的投影以及包围它们的投影的正矩形屏幕范围。假定对象已经过13.1.2节中的透视变换矩阵M的作用，因此，多边形在(x, y)平面上的正投影只须简单地将各顶点的z坐标分量忽略即得。在图13-4中，它们的屏幕范围并未重叠，因此它们的投影不需要进行相交测试。如果它们的屏幕范围重叠，那么可能会有两种情况（如图13-5所示）：投影也重叠（图13-5a）或投影不重叠（图13-5b）。在这两种情况中，都必须对投影是否重叠进行进一步判断。对图13-5b，通过判断得知，两部分投影其实并不相交，这表明，包围投影的屏幕范围相交并不总代表着投影本身相交。这种范围测试类似于裁剪算

445~446

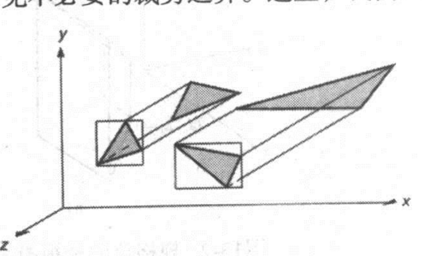
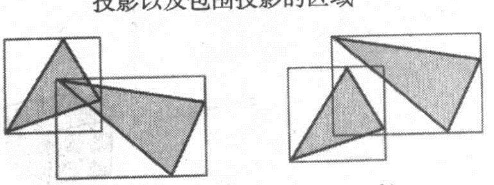


图13-4 两个对象、对象在（x, y）平面上的投影以及包围投影的区域



a) b)

图13-5 包围对象投影的区域。a)包围域与投影均重叠，b)包围域重叠而投影不重叠

法中的简易消除测试。

矩形区域测试也称为**包围盒**测试。这种区域也可用于像第7章中那样包围对象自身而不是它们的投影。这时的包围范围成为立体的，也称为**包围体**。另外，也有用于包围一维对象的，比如当我们需要判定两个对象在z轴上是否重叠。图13-6所示的便是这种情况：每一对象的最小与最大的z值之间的范围构成一个无限的包围域。满足下列条件时，两个对象在z轴上便无重叠：

$$z_{max2} < z_{min1} \quad 或 \quad z_{max1} < z_{min2} \tag{13-4}$$

在一维或多维度上利用最小与最大的边界进行比较也称为**极大极小界测试**（minmax testing）。极大极小界比较时，最复杂的部分在于确定范围本身。对于多边形（或其他完全被包含在由若干点定义的凸面体当中的对象）而言，重复计算所有点的坐标并记录坐标的最大与最小的值，便可计算出范围。

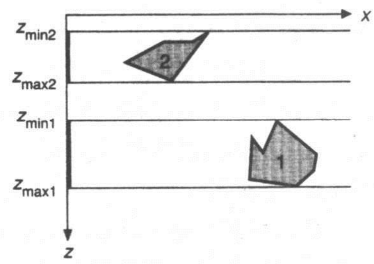


图13-6 用一维范围判定对象是否重叠

范围和包围体不仅可用于比较两个对象或它们的投影，也可用于判定投影线是否与对象相交。包括计算点和二维投影的交或向量和三维对象的交（见13.4节）。

到目前为止我们只讨论了极大极小范围，当然还存在其他的包围体形式。那么，什么样的包围体最好呢？毫无疑问，答案取决于用包围体本身进行测试所需的代价以及包围体防止其包含对象测试不产生相交的能力。Weghorst、Hooper与Greenberg[WEGH84]把包围体的选择作为一个对象相交测试的总代价函数T的极小化来处理。这可以表示为

$$T = bB + oO \tag{13-5}$$

其中b为包围体做相交测试的次数，B是执行包围体相交测试的代价，o是对象相交测试的次数（包围体实际相交的次数），O是对象做相交测试的代价。

由于只有当包围体实际相交时才会执行对象的相交测试，可知$o \leq b$。对特定对象与测试集合而言，O与b是常量，但B与o则随包围体的形状与尺寸大小发生变化。一个"较紧"的包围体，o会很小，但相应的B可能较大。包围体的有效性也可能受其包含对象的方位以及与其相交的对象的种类的影响。

13.1.4 背面消除

如果一个对象能够近似为一实多面体，那么它的多边形表面可以完全包围住它的体积。假定表面多边形的法向量指向多面体外部。如果多面体的内部完全位于前裁剪平面之前，那么那些表面法向量指向远离观察者一边的多边形将完全不可见（图13-7）。这些不可见的**背面**多边形应该在进一步的处理中被去掉，这种技术称为**背面消除**。类似地，非背面也可称为**正面**。

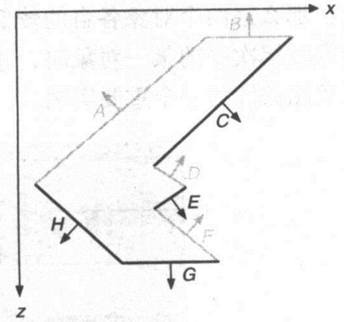


图13-7 背面消除。用灰色表示的背面多边形（A, B, D, F）被删除，而正面多边形（C, E, G, H）则保留下来

在眼睛坐标系中，判别背面的方法很简单。只需求出投影中心到多边形上任一点组成的向量与该多边形的表面法向量的点积。若为非负，则是背面。（严格地说，点积为正表明是背面。点积为零时表明多边形被视见为边）。假设已通过透视变换或需要在（x, y）平面上做正

交投影，投影方向为（0，0，-1）。这时，点积测试可简化为判断曲面法向量的z值，若为负，则为背面。如果环境中仅含有一个凸多面体，那么可见面判定工作也就是进行背面的剔除，否则可能存在一些正面，如图13-7中的C与E，它们是部分可见或完全不可见的。

如果在多面体中存在丢失的或裁剪过的正面，或多边形根本就不是多面体的一部分时，背面多边形需要做进一步的处理。如果不想删除，最简单的方法是把背面多边形当成正面，即翻转其法向量的方向。在PHIGSPLUS系统中，用户能够为面的每一侧指定完全独立的属性集合。

注意，一条穿过一个多面体的投影线一定穿过相同数目的背面多边形与正面多边形。于是，多面体投影中的点一定位于相同数目的背面与正面多边形的投影中。因此，在图像精度的可见面算法中，用背面消除能够省去一半的多边形计算量。平均来看，多面体中约有一半的多边形是背面的，因此，对对象精度的可见面判定算法而言，背面消除算法同样可以减少一半的运算量。（需要注意的是这只是指平均情况。例如，金字塔的塔基平面就有可能是惟一的背面或正面。）

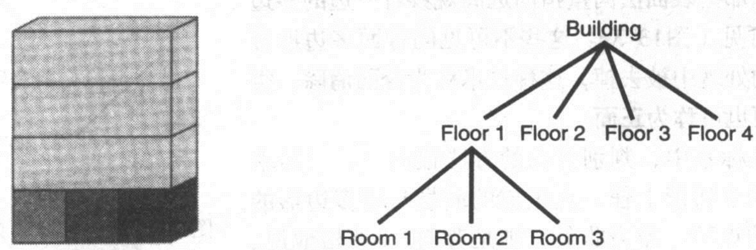
13.1.5 空间划分

空间划分（spatial partitioning）（也称为**空间细分**（spatial subdivision））的思想是将一个大的问题分解成一些小的问题，基本方法是在预处理阶段将对象或它们的投影分成空间上具有相关性的若干组。例如，我们可以将投影平面划分成规则的二维矩形网格，并求出每个对象的投影位于哪些网格空间。只有投影位于相同网格的多边形需要判断其是否重叠遮挡。这一技术
447~449 应用于[ENCA72；MAHN73；FRAN80；HEDG82]中。空间划分技术也可用于对空间对象添加规则的三维网格。这样，判定对象与投影线相交的过程可通过判断投影线与网格分区的交，再判断对象是否位于分区中来提高速度（13.4节）。

如果对象的空间分布非常不均匀，采用**适应性划分**会更有效，在适应性划分中每个分区的大小可变。一种可行的方法是递归地细分空间直到达到某一终止条件为止，如分区中对象少于某一分区的最大对象个数时细分终止[TAMM82]。10.6节中的四叉树、八叉树以及BSP树数据结构非常适合于这一要求。

13.1.6 层次结构

在第7章中已经讨论过，层次对联系各种结构以及不同对象的运动起着非常重要的作用。一种嵌入式的层次模型是将每个子节点作为其父节点的一部分。该结构能有效地限制可见面算法中对象比较的次数[CLAR76；RUBI80；WEGH84]。层次结构中某一层次的对象包含了它的所有子对象，可看成是其子对象的范围（如图13-8所示）。这时，如果某一层中的两个对象不相交，那么这两个对象各自的较低层次的对象之间就不需要进行相交测试。类似地，只有当投影线穿过层次中的某一对象时，才需要对该对象的子对象进行测试。这种层次结构的使用是体现对象相关性的一个重要实例。



450 图13-8 层次结构可用于限制对象比较的次数。仅当光线与Building和Floor1相交时才需要同Room1到Room3进行相交测试

在本章的其余部分，我们将讨论多种为可见面判定开发的算法。我们的重点是要计算出对象表面的哪些部分是可见的，至于表面颜色的判定需要留到第14章解决。在讲述每一个算法时，我们着重于其对多边形的应用，同时也指出何时能推广到处理其他对象。

13.2 z缓存算法

无论是从硬件还是软件的角度考虑，**z缓存**或**深度缓存**算法都是最简单的可见面判定算法。它最早由Catmull[CATM74]提出，属于图像精度算法。该算法需要一个帧缓存F用于存储每一点颜色值；还需要一个同样大小的**z缓存**（z-buffer）Z用来存储每一点的z值。z缓存初始化为0值，代表后裁剪平面的z值；帧缓区初始化为背景颜色。z缓存中能存储的最大z值代表前裁剪平面。多边形按任意顺序扫描转换到帧缓存。在扫描转换过程中，如果多边形对应到(x, y)的点比缓存中现有的点离视点更近，那么用新点的颜色与深度值替代旧值。程序13-1是z缓存算法的伪代码，读写缓存的过程在第3章中用的是WritePixel与ReadPixel过程，这里增加了WriteZ与ReadZ过程用于读写z缓存。

程序13-1 z缓存算法伪代码

```
void zBuffer()
{
int pz;                                        /* 像素坐标（x, y）处的多边形的z */
   for (y = 0; y < YMAX; y++) {
      for (x = 0; x < XMAX; x++) {
         WritePixel (x, y, BACKGROUND_VALUE);
         WriteZ (x, y, 0);
      }
   }
   for（每个多边形）{
      for（多边形投影中的每个像素）{
         pz = 在像素坐标(x, y)处的多边形的z值;
         if (pz >= ReadZ (x, y)) {                /* 新点并不是更远的 */
            WriteZ (x, y, pz);
            WritePixel( x, y, 在像素坐标(x, y)处的多边形的颜色);
         }
      }
   }
}
```

不需要预排序，也不需要进行对象与对象之间的比较。对确定的x与y而言，整个过程的运算量仅相当于对集合$\{Z_i(x, y), F_i(x, y)\}$的搜索以找到最大的Z_i。z缓存与帧缓存总是记录每一(x, y)处当前最大z值所对应的信息。因此，多边形按它们被处理的顺序显示在屏幕上，扫描转换每一多边形时一次处理一条扫描线到缓存中（见3.5节）。图13-9是两个多边形的叠加，两个多边形分别用不同的明暗度表示；z值也显示在图中。 451

利用深度相关性及多边形是平面的事实，可以简化扫描线上每点z值的计算。一般而言，要计算z值，必须求解平面方程$Ax + By + Cz + D = 0$：

$$z = \frac{-D - Ax - By}{C} \tag{13-6}$$

如果在(x, y)处式(13-6)求得z_1，则在$(x + \Delta x, y)$处z值为：

$$z_1 - \frac{A}{C}(\Delta x) \tag{13-7}$$

式中A/C为常量，则给定$z(x, y)$时，求$z(x+1, y)$只需要做一次减法，其中$\Delta x=1$。类似的增量计算可用于求下一扫描线的第一个z值，只需要对每一Δy减去B/C。另外，如果面未确定或多边形非平面（见9.1.2节），$z(x, y)$可通过多边形沿扫描线上的一对边处z值的插值获得（图13-10）。此处也可使用递增计算方法。这里当一点的可见性判定条件未满足时，不需要计算该像素的颜色。因此，当像素的明暗计算很费时时，先将对象粗略地按从前到后的顺序进行排序，可以有效地先显示最近的对象。z缓存算法并不要求对象由多边形构成。事实上，该算法最具吸引力的地方正是它能用于绘制任意的对象，只要该对象的投影的每一点处的明暗度及z值可以确定，从而不需要进行求交计算。

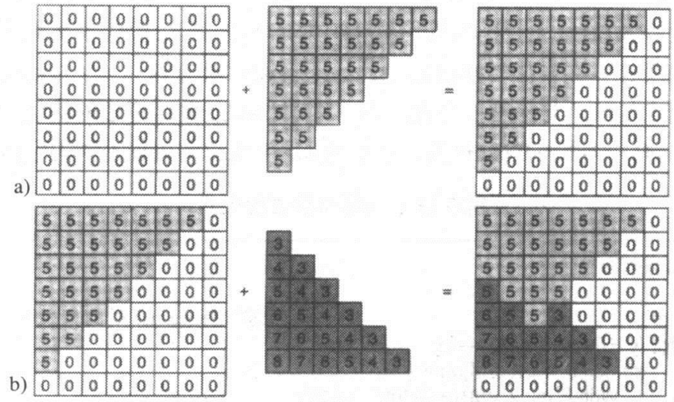


图13-9 z缓存。不同多边形覆盖的点用不同的灰度值表示，z值用数字标出。a)增加一个z为常量
452 的多边形到空的缓存中，b)增加另一个多边形与第一个相交

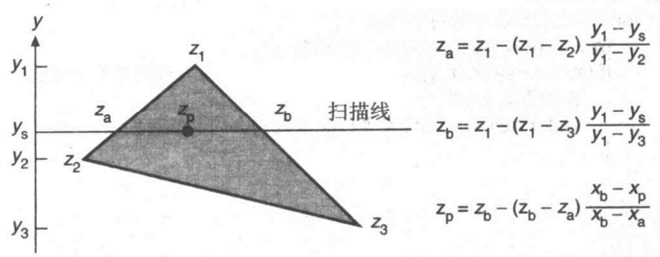


图13-10 多边形边与扫描线上z的插值。z_a由z_1与z_2之间插值获得；z_b由z_1与z_3之间插值获得；z_p由z_a与z_b之间插值获得

z缓存算法对x与y进行基本的排序，不需要进行比较运算。z排序在每一点处对包含该点的每个多边形只做一次比较。可见面计算需要的时间同对象中包含多边形的个数无关。因为，平均来看，当视见体中多边形个数增加时，每个多边形覆盖的点数却减少了。因此，搜索的每一集合的平均大小是固定的。当然，多边形增加带来的扫描转换量的增加也必须另外考虑。

尽管z缓存算法需要大量的存储空间，但实现起来却非常方便。当内存很紧张时，可以考虑将图像扫描转换成条状，只需要对处理的每一条图像准备足够的z缓存。缺点在于要对对象做多遍的处理。由于z缓存很简单且不需要附加的数据结构，随着内存造价的降低，许多基于z缓存的硬件和固件得到越来越广泛的应用。因为z缓存算法在图像精度进行操作，很容易产生走样的问题。A缓存算法[CARP84]用未加权面积采样的离散逼近算法来解决这一问题。

z缓存算法的硬件实现常采用16位到32位的整数值，而软件（有些硬件）实现采用浮点值。尽管16位的z缓存对许多CAD/CAM应用已经足够了，但对于毫米级细节定义的对象分布在几公

里范围的场景时就力不从心了。更糟的是，如果采用透视投影变换，变换除法带来的远距离z值的压缩效果会对深度排序和远距离对象的求交产生严重的影响。靠近视图平面时应该变换成不同的整数z值的两点，在距离很远时可能会变换成相同的z值（参见习题13.9和[HUGH89]）。

z缓存算法的精度有限，还会带来另一种走样问题。在绘制从不同端点出发的两共线边的公共部分时，扫描转换算法通常会作为两类不同的像素集来处理。这样，由于对z插值时可能会产生数值上的误差，这些边上相同的像素可能会得到略微不同的z值。这一点在绘制多面体的不同面的共享边时表现非常明显。在同一条边上，有些可见像素是某一多边形的一部分，而另一些像素却是另一相邻多边形的一部分。这个问题可以通过插入一些附加顶点以保证这些顶点位于公共部分的相同位置来解决。 453

在图像绘制出来以后，z缓存仍然有其存在的价值。因为可见面判定算法只用到这一种数据结构，可以将这种结构同图像保存在一起以便其他z值可计算的对象与其合并时使用。该算法也可以改写成当绘制选定的对象时z缓存内容不变。当z缓存用这种方式被屏蔽掉时，某一对象能够消除隐藏面并被绘制到一个独立的覆盖平面中（如果该对象是x与y的单值函数），并且能够在不影响z缓存内容的前提下被删除。这样，简单的对象，如规则网格等图形，能够在图像中的x，y与z方向移动，充当"3D指示"，并能与场景中的对象形成自然的遮挡与被遮挡关系。部分切除的效果也很容易实现，即根据z值是否位于某一切割平面之后来写z缓存与帧缓存。如果显示的对象对每一(x, y)有惟一的z值，那么z缓存的内容也可用于计算面积和体积。

Rossignac与Requicha[ROSS86]讨论了如何用z缓存算法处理CSG定义的对象。平面投影上的每一像素仅当它的z值更近且该平面构成CSG对象的一部分时才被写入。不同于在每一像素上只存储最近z值点，Atherton提出在z缓存中存入所有点的列表，这些点按z值排序，且存入每一点所属面的标识，构成**对象缓存**[ATHE81]。由后处理阶段确定图像如何显示。这样，许多特殊的效果（如透明、裁剪以及布尔集合运算）都可以通过处理每个像素的列表获得，而不用重新扫描转换这些对象。

13.3 扫描线算法

扫描线算法最早由Wylie、Romney、Evans和Erdahl[WYLI67],Bouknight[BOUK70a; BOUK70b]与Watkins[WATK70]等人提出，属于图像精度算法，每次处理一条扫描线而生成一幅图像。它的基本思想是3.5节中的多边形扫描转换算法的扩展，同时运用了各种形式的相关性，如扫描线相关性和边的相关性。差别在于我们不只处理一个多边形，而是多边形的集合。算法的第一步是对投影到视平面上的所有多边形的非水平边生成一个**边表**（ET）。同前面一样，水平边不予考虑。ET中的项基于每一边的较小的y坐标进行桶排序，在每个桶中的边又按照较低端点的x坐标递增的顺序排序。每一项包括： 454

1) 具有较小y坐标一端的x坐标。
2) 边的另一端的y坐标。
3) 从一条扫描线到另一条扫描线的x增量，Δx（Δx是边斜率的倒数）。
4) 多边形标识号，表明该边属于哪个多边形。

还需要建立一个**多边形表**（PT）。在PT表中对每个多边形除了其ID外，至少还包含以下内容：

1) 平面方程的系数。
2) 多边形的明暗度或颜色信息。
3) 一个in-out布尔量标志，初始化为false，在扫描线处理过程中要用到。

图13-11是两个三角形在(x, y)平面上的投影，隐藏边用虚线画出。这幅图经过排序的边表

（ET）中含有AB，AC，FD，FE，CB与DE。多边形表（PT）中含有ABC和DEF。

这里还要用到3.5节中的**活动边表**（AET）。它也是按 x 递增的顺序保存。图13-12中列出了ET、PT和AET项。当前，算法已经执行到扫描线$y = \alpha$，AET中依次序含有AB和AC。边从左到右进行处理。在处理AB时，首先转换多边形ABC的in-out标志。此时，标志变为true；于是，扫描是进入（"in"）多边形，要对多边形进行处理。现在，由于扫描是进入惟一的一个多边形ABC，它一定是可见的，因此，ABC的明暗度可以应用到AET表中从AB到AC的跨段上。这是一个利用跨段相关性的实例。在边AC上，ABC的标志变为false，因此扫描不再是进入（"in"）任何多边形。更进一步，因为AC是AET表中的最后一条边，扫描线处理完成。然后AET根据ET进行修改并再次按x排序（因为它的有些边可能已经交叉），继续处理下一条扫描线。

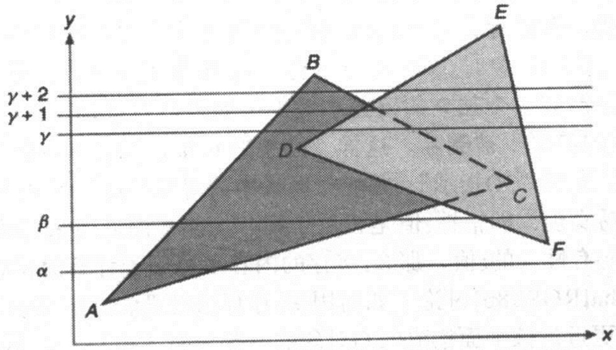
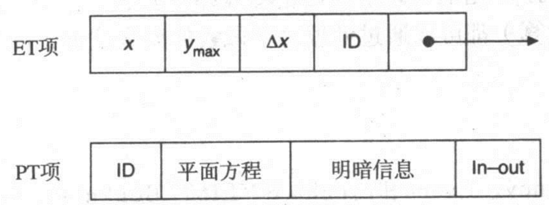


图13-11 用扫描线算法处理两个多边形



AET 内容

扫描线	项
α	AB AC
β	AB AC FD FE
$\gamma, \gamma+1$	AB DE CB FE
$\gamma+2$	AB CB DE FE

图13-12 扫描线算法中的ET、PT和AET

当遇到扫描线$y = \beta$时，AET表的顺序是AB，AC，FD和FE。处理和前面大同小异。扫描线上有两个多边形，但扫描一次仅进入一个多边形。

扫描线$y = \gamma$的情况较为复杂。进入ABC时它的标志变为true。ABC的明暗度用于直到下一条边DE的跨段。这时，DEF的标志也变为true，因此，扫描进入（"in"）两个多边形。（有必要保存一个in-out标志为true的多边形的列表以及列表中含有多边形的个数。）现在我们必须判定ABC和DEF中哪一个离视点更近。方法是根据两个多边形所在平面的方程求出z值，其中$y = \gamma$，x等于直线$y = \gamma$和边DE的交点处的x值。该x值是AET表中DE项的值。在这个例子中，DEF的z值较大，因而是可见的。因此，在假定不存在多边形穿透的情况下，直至边

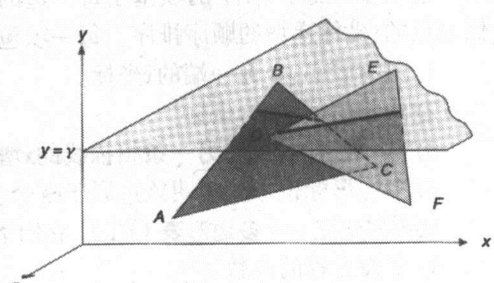


图13-13 多边形ABC与DEF和平面$y = \gamma$的交

455~456 CB的范围内应该用DEF的明暗度。到达边CB的这一点处，ABC的标志变为false，且扫描再次进

入惟一的多边形*DEF*，继续使用它的明暗度直到边*FE*。图13-13显示两个多边形与$y = \gamma$平面的关系；两条粗线是多边形与平面的交。

假设有一个大的多边形*GHIJ*位于*ABC*和*DEF*之后，如图13-14所示。那么当扫描线$y = \gamma$遇到边*CB*时，扫描仍然是进入（“in”）多边形*DEF*和*GHIJ*，这时要再次完成深度计算。但是，如果我们假设多边形不会互相穿透，这种计算就可以避免。这意味着，当扫描离开*ABC*时，*DEF*与*GHIJ*之间的深度关系不会改变，即*DEF*继续位于前面。因此，当扫描离开一个被遮挡多边形时，深度计算是不必要的，只有当离开一个遮挡多边形时，才需要计算深度。

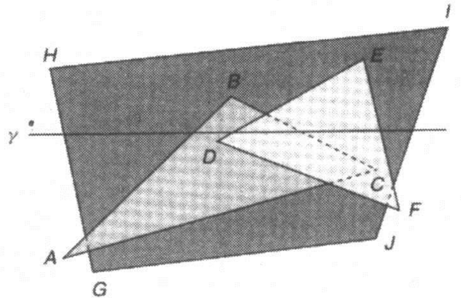


图13-14 三个非穿透性多边形。因为非穿透性多边形保持其相对的*z*的顺序，在扫描线γ离开被遮挡多边形*ABC*时不需要进行深度计算

要对穿透多边形使用该算法，需要做些变化。如图13-15所示，将*KLM*分割成*KLL'M'*与*L'MM'*。这里导入一条假边*M'L'*。另外，算法要经过适当的修改以找到扫描线上的穿透点。 457

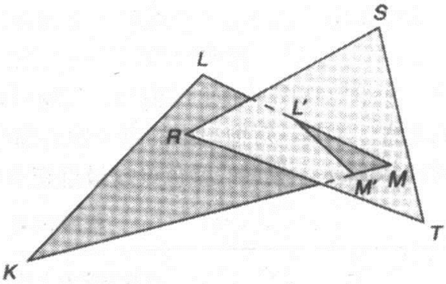


图13-15 多边形*KLM*在线*L'M'*处穿过多边形*RST*

这一算法的另一个修改是使用深度相关性。Romney注意到，假设多边形互不穿透时，如果一条扫描线上的位于AET表中的边与紧接着的前一条扫描线相同，且它们的顺序也相同，那么这条扫描线上各个部分的深度关系都没有发生改变，也就没有必要进行新的深度计算[ROMN69]。前一扫描线上可见跨段的记录也定义了当前扫描线的跨段。图13-11中的扫描线$y = \gamma$和$y = \gamma + 1$就是这种情况。两者都是从*AB*跨段到*DE*和从*DE*跨段到*FE*的部分可见。然而从$y = \gamma + 1$到$y = \gamma + 2$就不再满足深度相关性，因为边*DE*和*CB*在AET表中改变了次序（算法中必须作处理）。可见跨段也因此改变成从*AB*到*CB*和从*DE*到*FE*。Hamlin与Gear[HAML77]讨论了在AET表中边的次序改变的情况下仍能保持深度相关性的情形。

我们还没有讨论如何处理背景。最简单的方法是将帧缓存初始化成背景颜色，这样算法就只需要处理与边相交的扫描线。另一种办法是在场景中定义一个足够大的多边形，且令其比场景中所有其他的多边形都远，并平行于投影平面，然后给这个多边形赋予预期的明暗度。最后的办法是当扫描未进入任何多边形时直接在帧缓存中写入背景色。

尽管到目前为止我们处理的都是多边形，扫描线算法的使用可以非常广泛，包括更加一般性的面（13.5.3节中讨论的）。为实现这一点，ET和AET由**面表**（surface table）和**活动面表**（active-surface table）取代，这些表按面的(x, y)范围排序。一个面从面表移动到活动面表需要执行附加的操作。例如，可以将面分解成许多个逼近的多边形，这些多边形在扫描离开面的*y*轴范围时将被丢弃，这就省去了在整个绘制过程中维护所有面数据的麻烦。通用的扫描线算法的伪代码如程序13-2所示。Atherton[ATHE83]讨论了一种扫描线算法，用于绘制由结构实体几何（CSG）元素进行正则布尔集合运算生成的对象。

有一种扫描线算法因其简单而非常吸引人，它用*z*缓存来解决可见面问题[MYER75]。算法采用单扫描线帧缓存和*z*缓存对跨度进行累加，每一条新扫描线都要清除一次缓存。因为只需要一条扫描线的存储空间作为缓存，所以该算法非常适合于绘制高分辨率的图像。

程序13-2 一般扫描线算法的伪代码

```
向面表中加入面；
初始化活动面表；

for（每条扫描线）{
   更新活动面表；

   for（扫描线上的第个像素）{
      判定活动面表中报影到像素的面；
      找出最近的这种面；
      确定在该像素处最近面的明暗度；
   }
}
```

13.4 可见面光线跟踪

光线跟踪（也称为**光线投射**）通过跟踪从观察者眼睛到场景中物体的想像中的光线来确定面的可见性[⊖]。这是一种典型的图像精度算法（本章开始时讨论过）。选定一个投影中心（观察者的眼睛）和给定视图平面上的窗口，该窗口被划分为规则网格，网格的每一元素对应于要求分辨率下的像素。于是，对窗口中的每一像素都有一条视线从投影中心出发，穿过像素中心到达场景中，如图13-16所示。像素的颜色设成最近交点处对象的颜色。简单光线跟踪的伪代码见程序13-3。

程序13-3 简单光线跟踪的伪代码

```
在视图平面上选择投影中心和窗口；
for（图像中的每条扫描线）{
   for（扫描线上的每个像素）{
      确定从投影中心穿过像素的线；
      for（场景中的每个对象）{
         if（对象被交并且最近）
            记录交线和对象名；
      }
      设置像素的颜色为最近对象交线的颜色；
   }
}
```

光线跟踪算法最早由Appel[APPE68]以及Goldstein与Nagel[MAGI68；GOLD71]提出。Appel用光线的一个稀疏网格判定场景明暗度，包括确定一点是否在阴影中。Goldstein与Nagel最初用他们的算法模拟弹道发射与核粒子的轨迹，后来才应用于图形学中。Appel最早研究光线跟踪计算阴影，而Goldstein与Nagel最先用光线跟踪求布尔集合运算。Whitted[WHIT80]与Kay[KAY79a]将光线跟踪算法扩展到计算高光反射与折射。我们将在14.7节中讨论阴影、反射与折射等人们所熟知的光线跟踪适用的领域，我们还将给出一个合成了可见面判定与明暗处理的完整的光线跟踪递归算法。在本节，我们只讨论光线跟踪在可见面判定方面的应用。

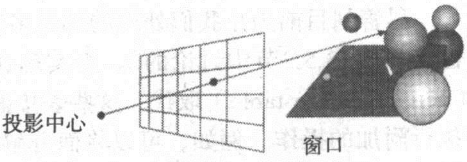


图13-16 光线从投影中心发射经过窗口映射到每一个像素，判定最近的相交对象

⊖ 尽管光线投射与光线跟踪常被混同使用，但有时光线投射用于专指本节的可见面算法，而光线跟踪指的是14.7节中的递归算法。

13.4.1 相交计算

光线跟踪算法的核心是计算光线与场景中对象的交点。为实现这一点，这里采用与第3章中相同的向量的参数表示。从(x_0, y_0, z_0)到(x_1, y_1, z_1)的光线上的每一点(x, y, z)由参数t定义如下：

$$x = x_0 + t\,(x_1 - x_0), \qquad y = y_0 + t\,(y_1 - y_0), \qquad z = z_0 + t\,(z_1 - z_0) \tag{13-8}$$

为方便起见，我们定义Δx，Δy与Δz为

$$\Delta x = x_1 - x_0, \qquad \Delta y = y_1 - y_0, \qquad \Delta z = z_1 - z_0 \tag{13-9}$$

因此，

$$x = x_0 + t\,\Delta x, \qquad y = y_0 + t\,\Delta y, \qquad z = z_0 + t\,\Delta z \tag{13-10}$$

如果(x_0, y_0, z_0)为投影中心，(x_1, y_1, z_1)为窗口中像素的中心，则t从0变化到1代表这两点之间的点；t为负值时代表投影中心之后的点，而t大于1时对应于窗口远离投影中心一边的点。对每一类对象我们需要找到一种表示方式，以方便地求出交点处的t值。最简单的对象是球体，这就是为什么在典型的光线跟踪图像中存在大量球体的原因！以(a, b, c)为球心，半径为r的球体用如下方程表示：

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2 \tag{13-11}$$

展开式(13-11)，并将式(13-10)中的x, y, z值代入，可得：

$$x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2 \tag{13-12}$$

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 + (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2 + (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2 \tag{13-13}$$ 458~460

$$\begin{aligned} &x_0^2 + 2x_0\Delta xt + \Delta x^2t^2 - 2ax_0 - 2a\Delta xt + a^2 \\ &+ y_0^2 + 2y_0\,\Delta yt + \Delta y^2t^2 - 2by_0 - 2b\Delta yt + b^2 \\ &+ z_0^2 + 2z_0\,\Delta zt + \Delta z^2t^2 - 2cz_0 - 2c\Delta zt + c^2 = r^2 \end{aligned} \tag{13-14}$$

多次合并后可得：

$$\begin{aligned} &(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] \\ &+ (x_0^2 - 2ax_0 + a^2 + y_0^2 - 2by_0 + b^2 + z_0^2 - 2cz_0 + c^2) - r^2 = 0 \end{aligned} \tag{13-15}$$

$$\begin{aligned} &(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x\,(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] \\ &+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0 \end{aligned} \tag{13-16}$$

式(13-16)是关于t的二次方程，其系数全是可从球与光线方程中导出的常量。因此可用二次方程公式求解。如果没有实根，那么光线与球不相交；如果有一个实根，则光线与球相切于一点；如果有两个根，则相交于两点，其中得到最小正t值的是最近点。将光线规格化使得从(x_0, y_0, z_0)到(x_1, y_1, z_1)之间的距离为1将可简化计算。这样其t值度量的将是WC单位表示的距离。同时，相交运算亦得到了简化，因为式(13-16)中t^2的系数为1。我们可以用类似的方法求得光线与第9章中介绍的普通二次曲面的交点。

正如在第14章中我们将会看到的，必须确定交点处曲面的法向量以对曲面进行明暗处理。这一操作对球体而言非常简单，因为球面上未规格化的法向量是从球心指向交点的向量：设球心为(a, b, c)，则交点(x, y, z)处的表面法向量为$((x - a)/r, (y - b)/r, (z - c)/r)$。

求解光线与多边形的交点相对比较困难。一般的方法是先确定光线是否与多边形所在平面相交，然后判断交点是否位于多边形内部。平面的方程可表示为

$$Ax + By + Cz + D = 0 \tag{13-17}$$

代入式(13-10)得

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0 \tag{13-18}$$

$$t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0 \tag{13-19}$$

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)} \tag{13-20}$$

如果式(13-20)中分母为0，那么光线与平面平行没有交点。一种判定交点是否位于多边形
461 内部的简单方法是将多边形与交点正投影到定义坐标系的三个平面之一（如图13-17所示）。要获得最精确的结果，应该选取有最大投影的方向为轴，这对应于多边形的平面方程中系数绝对值最大的坐标。对多边形顶点和该点的坐标做正投影。于是，该点的多边形包含测试就可完全在二维中完成，算法参见7.11.2节。

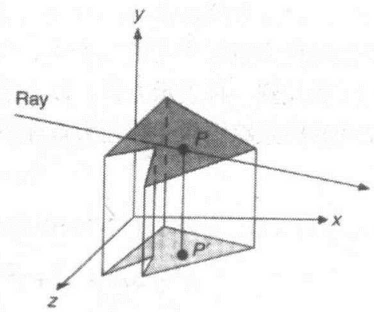


图13-17　光线与多边形是否相交的判定。光线与多边形平面的交点p被投影到其中一个坐标系平面上，判断投影点p'是否位于投影多边形内部

同z缓存算法类似的是，光线跟踪也只在投影线和对象之间求交，而不需要直接判断场景中两个对象之间的交。z缓存算法将对象近似成与对象相交的投影线上的一系列z值；而光线跟踪算法将对象近似成与场景相交的投影线上的一系列的交点。对于一类新的对象，我们只需要为其写一个扫描转换和计算z值的过程便可实现z缓存算法的扩充；同样，我们只需要写一个光线相交的过程便可实现可见面光线跟踪算法中新对象的扩充。在两种情况下，都必须有一个计算曲面法向量的过程，用于进行明暗处理。已经为代数曲面[HANR83]、参数曲面[KAJI82；SEDE84；TOTH85；JOY86]开发了许多相交和曲面法向量算法。这些算法的概述参见[HAIN89；HANR89]。

13.4.2 可见面光线跟踪算法的效率

在每一像素，z缓存算法只计算投影到该像素的对象的信息，同时利用相关性。比较而言，
462 我们讨论的可见面光线跟踪算法简单而运算量大，需要对从眼睛发出的每一条光线与场景中的每一个对象求交。一幅含100个对象的1024 × 1024的图像就需要100M次的相交计算。无怪乎Whitted发现75%~95%的系统时间都花在了求交计算上[WHIT80]。因此，我们有必要讨论一下，如何能够加速每一步的相交计算或尽可能地省略掉。正如我们将在14.7节中看到的，递归光线跟踪算法跟踪来自交点的附加光线以判定像素的明暗度。因此，13.1节中的许多技术，如透视投影和背面消除等，并不总是有用的，因为并非所有的光线都从同一个投影中心发出。

1. 优化相交计算

在对象-光线求交的方程中有许多项中包含表达式，这些表达式要么对整幅图像是常量，要么对某一特定光线是常量。这些表达式可以预先计算出来，如多边形到平面的正投影等。经过认真的比较和数学上的研究，我们发现有可能提出快速的求交计算；即使是13.4.1节中的最简单的球体求交公式也有改进的余地[HAIN89]。如果光线变换到与z轴同方向时，相同变换也可作用到每个候选对象上，这样，任何的相交都将发生在$x = y = 0$处。这就简化了求交计算并允许最近的对象通过z排序得到。在明暗处理计算时可通过反向变换将交点再变换回来。

包围体提供了另一种加速求交运算的途径。一个求交相对复杂的对象，对其包围体求交就要相对简单得多，这些包围体可以是球体[WHIT80]、椭球体[BOUV85]或长方体[RUBI80；

TOTH85]。光线不能与包围体相交时，包围体中的对象就不需要进行测试。

2. 层次结构

尽管包围体本身并不能确定次序或相交测试的频率，包围体可以组织成嵌套的层次结构，对象位于叶节点与内部节点，每个节点又包围它的子节点[RUBI80；WEGH84；KAY86]。如果包围体的父节点不与光线相交，那么该包围体也不会与光线相交。根据这一点，当相交测试从根开始时，许多级别的分支（含有许多对象）都将被简单地拒绝。

3. 空间划分

包围体层次结构将对象自底向上地组织在一起；而空间划分方法则是将空间进行自顶向下的划分。首先计算场景的包围盒。有一种方法是将包围盒分成等距离的规则网格（如图13-18所示）。每一个分区都对应一个对象列表，这些对象要么完全位于该分区中，要么仅有一部分在里面。填充列表时将每个对象归入包含它的一个或多个分区中。如图13-19所示的2D情况中，光线只与它穿过的分区中的对象相交。另外，分区按照光线穿过的顺序进行处理，这样，只要找到一个有相交的分区，就不需要再处理剩余的分区。要注意的是，我们必须考虑该分区中所有其他的对象，以确定哪个交点最近。由于分区是均匀网格，沿光线方向的每一个后续分区的计算可以采用3.2.2节中介绍的三维画线算法。略加修改以生成光线经过分区的列表[FUJI85；AMAN87]。

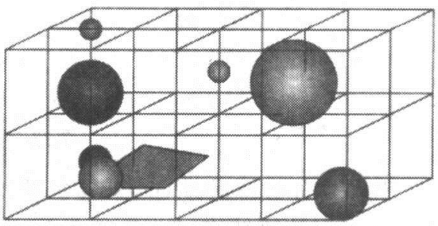


图13-18 场景划分成大小相等的均匀网格

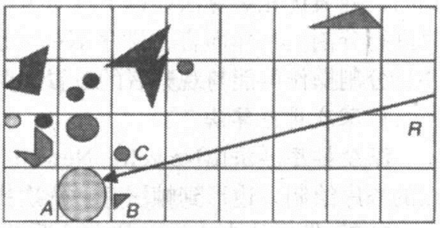


图13-19 空间划分。光线R只可能与A，B和C相交，因为它穿过的其他分区都是空的

如果光线与某分区中的对象相交，还需要判断交点是否位于该分区中；有可能交点位于很远的另一个分区中，而另一个对象则有更近的一个交点。例如，在图13-20中，对象B尽管位于分区2中，却相交于分区3。必须继续遍历各个分区直到在当前分区中找到一个交点，在此例中是分区3中的A。为了避免光线与对象在多个分区中重复求交，可以在首次处理对象时将交点和光线的ID随对象缓存起来。

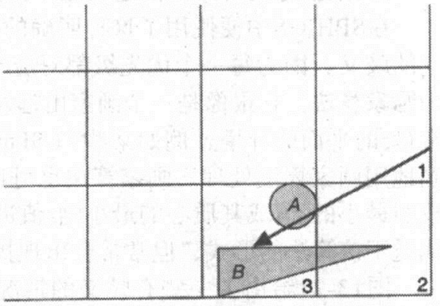


图13-20 对象可能在另一个体元中相交

Dippé与Swensen[DIPP84]研究了自适应划分算法，生成大小不等的分区。另一种自适应空间划分方法利用八叉树来划分场景[GLAS84]。算法中用10.6.3节中描述的八叉树寻邻区算法来确定沿光线的下一个分区[SAME89b]。八叉树和其他的层次空间划分方法都可以看成是层次划分的特例，在这种层次结构中，任一节点的子节点之间互不相交。因为这些方法允许采用自适应划分，划分策略的选择对分区中对象的数量和对象相交的运算量很敏感。这在不规范、不均匀分布的场景的处理中非常有利。

13.5 其他方法

13.5.1 列表优先级算法

列表优先级算法确定对象的可见性排序以保证对象按这一顺序绘制时能得到正确的图像结

果。例如，如果对象之间在z方向上不存在重叠的情况，那么，我们只需要将对象按z值的增加进行排序，并按照这一顺序进行绘制。较近的多边形会覆盖较远的多边形，从而使近的对象遮挡住较远的对象。如果在z方向上存在重叠的现象，我们仍然有可能判断出正确的顺序，如图13-21a所示。如果对象之间循环重叠，如图13-21b和图13-21c所示，或互相穿透，那么就没有正确的顺序。这时，就有必要将一个对象分割成几部分以使得线性排序成为可能。

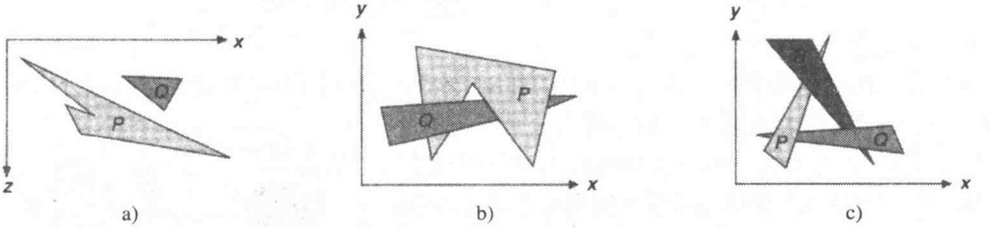


图13-21 z方向上多边形重叠的例子

列表优先级算法是对象精度算法与图像精度算法的混合。深度比较与对象分割都是对象精度的。只有扫描转换（它覆盖已绘制对象的像素依赖于图形设备的性能）是图像精度的算法。因为排序对象的列表是以对象精度生成，所以可以以任意的分辨率重复显示。正如我们将要看到的，列表优先级算法在以下几个方面有所不同：如何确定顺序，哪些对象需要被分割以及何时进行分割。具体而言，排序不一定是在z方向上，有些对象既没有循环重叠也没有互相穿透，并且分割操作可能与观察者的位置无关。

1. *深度排序算法*

*深度排序算法*由Newell、Newell与Sancha[NEWE72]提出，其基本思想是按照逐步靠近视点的顺序绘制多边形到帧缓存。依次执行以下三步：

1) 按z坐标从小（远）到大（近）对所有的多边形排序。

2) 解决所有可能出现的排序上的歧义，这些歧义可能由z方向上的多边形重叠导致，必要时对多边形进行分割。

3) 按z坐标的升序（由远及近）扫描转换每一个多边形。

在SPHIGS中便使用了这种明确的优先级系统，它取代了最小z值的表示法，不再会有深度上的歧义，因为每一个优先级都对应于一个不同的常量z的平面。深度排序的简化形式常常称为**画家算法**，它很像是一个画家由远及近地绘画的过程，近的对象覆盖在远的上面。对象位于常量z的平面的环境，例如某些VLSI布局、制图以及窗口管理，称为是二维半，它们都可以正确地用画家算法处理。画家算法也可用于多边形并不位于常量z平面中的场景，此时的多边形按照最小的z值或其质心的最小的z值进行排序，上述的第二步可省略掉。尽管可以将场景构造成适于该算法的形式，但常常会出现排序不正确的情况。

图13-21给出了一些有歧义的情况，需要执行上述的第二步来加以解决。具体步骤如下：假设当前排序列表的远端多边形为P。在该多边形扫描转换到帧缓存之前，必须与所有的在z方向上与P有重迭的多边形（设为Q）进行测试，以保证P不能遮挡Q从而使P能够在Q之前被写入。最多执行5次测试，一次比一次复杂度高。一遍成功后，即P不能遮挡Q时，下一个多边形Q继续进行测试。如果所有的多边形都通过了测试，那么P就开始扫描转换，列表中的下一多边形成为新的P。这5次测试如下：

1) 多边形的x方向是否重叠？

2) 多边形的y方向是否重叠？

3) P是否完全位于Q平面的远离视点的一边？（图13-21a不成立，图13-22a成立。）

4) Q是否完全位于P平面与视点相同的一边？（图13-21a不成立，图13-22b成立。） [463~466]

5) 多边形到(x, y)平面的投影是否不重叠？（这一点可以通过对多边形的边之间进行比较来判定。）

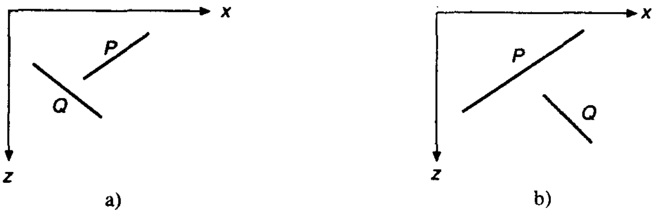


图13-22 可能的多边形方向。a) 测试3为真，b) 测试3为假，而测试4为真

如果所有5项测试都失败，可假定此时P遮挡Q，因此需要测试Q是否能够在P之前扫描转换。测试1、2和5并不需要重复执行，但测试3和4中的多边形要做一次交换：

3') Q是否完全位于P平面的远离视点的一边？

4') P是否完全位于Q平面与视点相同的一边？

在图13-21a中，测试3'为真。因此，可以将Q移动到列表的顶端使之成为新的P。而在图13-21b中，测试仍然不能得出结论。事实上，P与Q仍然不能按照正确的顺序完成扫描转换。这时，P或Q中必须有一个被对方的平面所分割（见3.11节中有关多边形裁剪，将裁剪边作为裁剪平面）。初始的未分割多边形被舍弃，分割后的碎片按照正确的z排序插入列表，算法同前面一样继续进行。

图13-21c是一种更复杂的情形。P，Q与R中任一个多边形都可以移动到列表的顶端，使其与另一多边形确定正确的顺序，但不能保证同另两个都形成正确的顺序。这将产生一个无限的循环。为了避免循环处理，必须对移动到列表顶端的多边形进行标记。这样，当最初的5次测试失败且当前的多边形Q被标记时，测试3'与4'就不需要执行，而是将P或Q做分割（就好像测试3'与4'都失败了），再将分割后的碎片重新插入。

2. 二元空间划分树

二元空间划分树（BSP树）算法由Fuchs、Kedem和Naylor[FUCH80；FUCH83]提出，是一种非常有效的计算静态三维多边形集合之间可见性关系的算法，适用于视点任意变化的情形。该算法以费时费空间的预处理过程为代价，获得一个线性的显示算法，每次视点变化时执行一次该线性算法。该算法非常适用于视点变化，而场景对象不变的情形。

BSP树算法的基本思想是基于这样一个事实：对于一个多边形而言，如果位于它的远离视点一边的所有多边形最先被显示；然后显示它自身；最后显示它的位于视点一边的多边形，那么该多边形一定能够保证其扫描转换的正确性（即不会产生与其他多边形不正确的重叠情况）。我们需要保证这一点对每一个多边形都成立。

BSP树算法通过构造一棵多边形的二叉树（即BSP树）使多边形之间的顺序非常容易判定。BSP树的根是从所有待显示的多边形中选择出来的，无论选择哪一个多边形，算法都能正确运行。根多边形用于将整个环境划分成两个半空间。一个半空间包含所有位于根多边形之前的多边形，即与它的表面法向量对应的一边；另一半空间包含根多边形之后的多边形。同时位于前半空间与后半空间的多边形被根多边形所在平面分割，所得的两部分分别归入相应的半空间。在前后半空间中又可以分别选出一个多边形作为根节点的前后子节点，每一个子节点再递归地用于划分各自半空间中的剩余多边形，依此类推。当每个节点都只包含一个多边形时，算法结束。 [467]

显然，对于给定的任意视点，我们都可以通过按一定次序遍历BSP树的方法获得依优先级正确排序的多边形列表。我们可以从根多边形看起，它将其余的多边形分成两个集合，每一个都完全位于该多边形所在平面的一边。因此，算法只需要保证各集合之间显示的相对次序正确

即可，即两个集合的多边形之间互不影响，且根多边形能够正确地显示并保持与其他集合之间的正确次序。如果视点位于根多边形的前半空间，算法必须先显示后半空间的多边形（那些可能被根遮挡的多边形），然后显示根，最后显示前半空间的多边形（可能遮挡根的多边形）。反之，如果视点位于根多边形的后半空间，算法必须先显示前半空间的多边形，然后显示根，最后显示后半空间的多边形。若多边形的法向量垂直于视线，则任一次序显示都可满足。当视点位于某一多边形的后半空间时，可以不显示这个多边形以实现背面的消除。根节点的每一个子节点也都递归地得到处理。建立BSP树的伪代码以及显示树的伪代码在[FOLE90]中给出。

与深度排序算法类似，BSP树算法中的插入和排序完全是对象精度的，同时也依赖于光栅设备的图像精度的填充能力。不同的是，所有多边形的分割都在预处理阶段完成，并且只有当环境发生改变时才需要重新执行。值得注意的是多边形分割的次数可能比深度排序算法要多。

列表优先级算法允许使用硬件的多边形扫描转换器，从而在速度上大大优于在每点判断z值的软件算法。深度排序算法和BSP树算法按由后向前的顺序显示多边形，可能会遮挡住较远的对象，所以，和z缓存算法类似，每一个像素的明暗计算可能执行多次。反之，多边形也能按从前往后的顺序显示，这时，多边形上的像素只有在尚未写入时才被显示。

在用列表优先级算法实现隐藏线消除时，必须特别注意由细分过程带来的新边。如果这些边和初始的多边形边一样进行扫描转换的话，就会出现不希望看见的效果，因此必须对其做上标记，以便不参加扫描转换。

13.5.2 区域细分算法

区域细分算法与空间划分算法一样，都遵循分而治之的思想，只不过区域细分是针对投影平面的。投影图像的区域作为考察对象。如果很容易确定区域中的多边形可见，便显示这些多边形，否则，将该区域划分为更小的区域，递归地执行上述策略。区域变小后，会有较少的多边形覆盖

468 每一个区域，直到最后能够作出决定。这种方法利用的是**区域相关性**，即一幅图像的足够小的区域最终会包含在至多一个可见多边形中。

Warnock算法

Warnock[WARN69]提出的区域细分算法将每一区域划分成四个相等的方块。在递归细分过程的每一步骤中，每个多边形的投影和感兴趣区域的关系可能为以下四种关系之一（见图13-23）：

1) **包围多边形**完全包含感兴趣的（阴影）区域（图13-23a）。
2) **相交多边形**与该区域相交（图13-23b）。
3) **内含多边形**完全位于该区域中（图13-23c）。
4) **分离多边形**完全位于该区域外（图13-23d）。

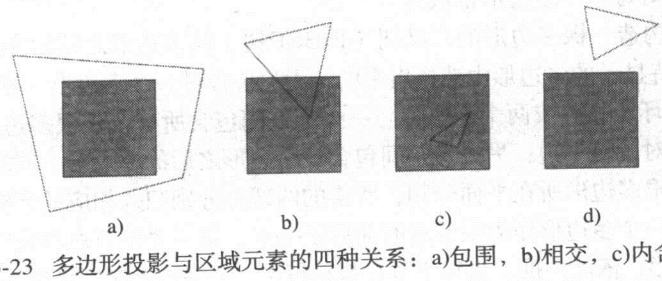


图13-23 多边形投影与区域元素的四种关系：a)包围，b)相交，c)内含，d)分离

分离多边形显然对感兴趣区域没有影响。相交多边形位于区域外的部分显然也没有影响，而位于区域内的部分和内含多边形一样，处理方法也一样。

在以下的四种情况下，区域中的可见性很容易判定，不需要再细分下去：

1) 所有的多边形都脱离区域，区域中只需要显示背景色。

2) 只有一个相交或内含多边形。该区域首先填充背景色，然后扫描转换多边形含在区域中的部分。

3) 含单个的包围多边形，但没有相交或内含多边形。该区域用包围多边形的颜色填充。

4) 多于一个多边形相交、内含于或包围该区域，但有一个包围多边形位于所有其他多边形之前。在所有多边形之前的判别方法是计算所有平面在区域四个角处的z坐标。如果存在一个包围多 469
边形的 z 坐标比所有其他多边形大（近于视点），那么整个区域就可用该包围多边形的颜色填充。

情况1、2和3很容易理解。情况4的进一步说明见图13-24。在图13-24a中，包围多边形的四个交都比其余的交要近于视点（它处在 + z轴无穷远）。因此，整个区域都被填充上包围多边形的颜色。在图13-24b中，无法做出判定，尽管此时似乎包围多边形在相交多边形之前，但在左边相交多边形平面位于包围多边形之前。在深度排序算法中，如果相交多边形完全位于包围多边形的一边且包围多边形远离视点，就不需要再进一步的细分。而Warnock算法总是要对区域细分下去以简化问题的复杂度。细分之后，只有内含多边形和相交多边形需要重新考虑：原始区域的包围多边形和分离多边形仍然是其子区域的包围多边形和分离多边形。

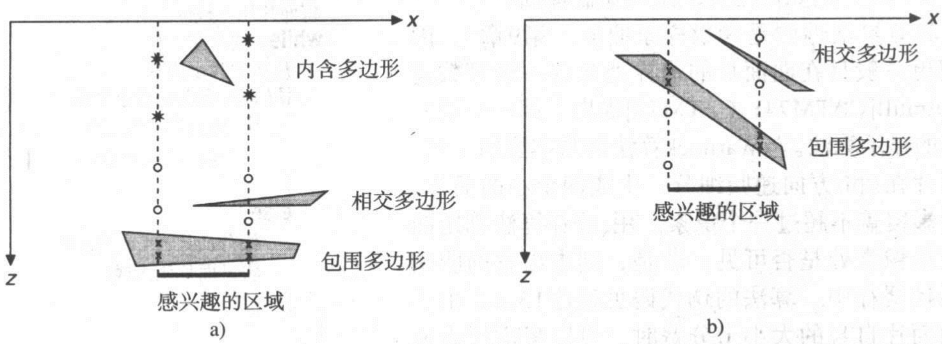


图13-24 递归细分中情况4的两个例子。a)包围多边形在感兴趣区域的所有角都离视点最近；b)相交多边形平面在区域左边离视点最近，×标记出包围多边形平面的交，○标记出相交多边形平面的交，*标记出内含多边形平面的交

到目前为止，除了背景的扫描转换操作与四种情况下的多边形的分割之外，该算法都操作在对象精度。然而其图像精度的扫描转换操作也可由对象精度操作替代，从而输出可见面的精确表示：一块区域的大小（情况1、3和4）或裁剪到该区域的单个多边形以及它相对于区域的补集，代表背景的可见部分（情况2）。那么，如果不是这 470
四种情况之一怎么办？一种方法是当达到显示设备的分辨率时便停止细分。这样，在一台1024×1024的光栅显示设备上，最多需要10次细分。如果已经执行了最大数目的细分，而四种情况都还未出现，那么需要计算在像素大小的不可见区域中心的所有相关多边形的深度。有最近z坐标的多边形用于确定该区域的明暗度。另外，出于反走样的考虑，可以继续对像素点进行细分，以分得子像素区域的大小作权值，从而求出该点的颜色。这些图像精度的操作只有在不满足简单情况时才执行，但正是这些图像精度的操作使得该算法成为图像精度算法。

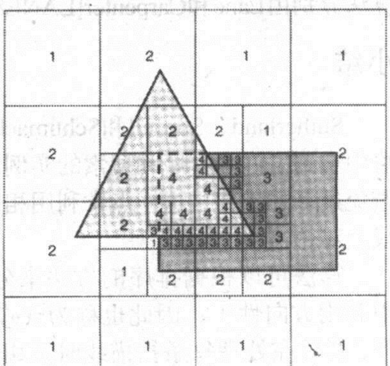


图13-25 区域细分为方块

图13-25是一个简单的场景和对场景的细分。每一细

分区域中的数字代表四种情况之一；在未标数字的区域，四种情况均不满足。可以将该算法同用四叉树做二维空间划分的算法（10.6.3节）进行比较。

13.5.3 曲面算法

到目前为止，我们所讨论的算法，除了z缓存算法外，都只适用于多边形面定义的对象。第9章中的曲面对象只有用许多小的面片逼近处理后，才能使用适于多边形的算法。尽管利用多边形可以逼近曲面，但我们常常更希望直接用扫描转换去处理曲面，一方面可以消除多边形
471 带来的视觉上的不适，另一方面可以避免逼近带来的额外存储开销。

9.4节中讨论的二次曲面是计算机图形学中常用的形式。二次曲面的可见面算法由Weiss[WEIS66]、Woon[WOON71]、Mahl[MAHL72]、Levin[LEVI76]和Sarraga[SARR83]开发。他们都找到了两个二次曲面求交的方法，给出了一个x, y, z的四阶方程，其解可通过数值方法求得。Levin用参数化相交曲线的方法将问题简化为二阶。球面是一种特殊的二次曲面，处理起来相对容易些，因而更多地得到应用。分子就常常显示成许多有色球体的集合（见彩图22）。人们开发了许多分子显示算法[KNOW77；STAU78；MAX79；PORT79；FRAN81；MAX84]。在13.4节中将讨论如何用光线跟踪绘制球面。

更具有灵活性的是参数样条曲面（第9章），因为它们更一般且在曲面片的边界处允许一阶导数连续。Catmull[CATM74；CATM75]提出了第一个双三次曲面的显示算法。与Warnock算法的基本思想一致，对曲面片在s和t方向递归细分，生成四个小曲面片，直至投影覆盖不超过一个像素。用z缓存算法判定曲面片在该像素处是否可见。若是，则求出它的明暗并放入帧缓存中。算法的伪代码见程序13-4。由于确定曲面片自身的大小十分费时，可以考虑用由面片四个角顶点定义的四边形来代替操作。

程序13-4 Catmull递归细分算法的伪代码

```
for（每个面片）{
  将面片压入栈;
  while（栈非空）{
    从栈中弹出面片;
    if（面片覆盖的像素个数≤1）{
      if（曲面片的像素在z方向上更近）
        确定明暗并画出;
    }
    else {
      细分面片为4个子面片;
      将子面片压入栈;
    }
  }
}
```

另一种是基于双三次曲面片自适应细分的方法，它确定了一个认为面足够平的容限，当曲面满足这个值时，划分中止。这个容限取决于显示设备的分辨率和被分区域相对于投影平面的方向。这就大大减少了不必要的细分。曲面片在一个方向足够平时，就只需要在另一个方向进行细分。细分程度足够细之后，曲面片可看成是四边形。用每个面片的四个角定义一个小的多边形区域，直接用扫描线算法处理，此时，允许多边形和双三次曲面很自然地混合在一起。使用这一基本思想的
472 算法分别由Lane和Carpenter[LANE80]以及Clark[CLAR79]提出，这些算法在[FOLE90]中描述。

小结

Sutherland、Sproull和Schumacker[SUTH74a]强调可见面判定的核心是排序。确实，我们在算法中看到了许多排序和搜索的实例。有效的排序对有效的可见面判定非常重要。同样重要的是要避免过多的排序，这一点常利用相关性来实现。例如，扫描线算法用扫描线相关性消除每一扫描线上对x的完全排序。

算法可以根据排序的方法来分类。深度排序算法先按z，然后x，然后y来排序（测试1和2中利用方向性）；因此也称为zxy算法。扫描线算法对y排序（桶排序），然后对x（先为插入顺序，然后在处理每条扫描线时采用冒泡排序），最后在z方向搜索距离视点最近的多边形；因此，这是yxz算法。Warnock的算法对x和y做平行的排序，然后搜索z方向，因此是一种$(xy)z$算法（括号表示维度的组合）。z缓存算法除了z方向外，没有明确的排序和搜索；称为(xyz)算法。

Sancha认为排序时依照的顺序是无关紧要的：沿某一特定轴方向排序并不比沿另一方向有实质性的优势，至少在原则上，平均来看，对象在三个方向上都具有相同的复杂度[SUTH74a]。另一方面，类似于Hollywood集合的图形场景可以通过一定的构造使其从某一特定视点看会更好，也会使其在某一方向具有更高的复杂度。尽管我们可以假定对象具有对称的复杂度，但并非所有的算法都具有相同的效率：它们的不同在于是否能有效地利用相关性来避免多余的排序和计算以及是否能在空间-时间上给出最佳的平衡。在[SUTH74a, Table VII]中给出了四种基本算法的性能评估比较，概括起来如表13-1所示。作者提出，既然都只是估计，小的差别可以忽略，但“我们希望对不同的算法有一个数量级的比较，以期能对不同算法的效率有一定的了解”[SUTH74a，p.52]。

表13-1 四种可见面判定算法的相对性能评估

	场景中多边形面的个数		
算法	100	2 500	60 000
深度排序	1①	10	507
z缓存	54	54	54
扫描线	5	21	100
Warnock区域划分	11	64	307

① 经规格化后将此项作为一个单位。 473

深度排序算法对少量的多边形有效，因为对于判断一个多边形是否能扫描转换，简单的重叠测试就足够了。多边形增多时，就需要更复杂的测试，也可能需要对多边形进行分割。z缓存算法的性能是常量，因为，当场景中的多边形数目增加时，单个多边形覆盖的像素点数却减少了。另一方面，它的内存需求较大。Warnock的区域划分算法的每一步测试和计算都较复杂，因此实现起来通常比其他方法要慢。除了这些非形式化的评估外，还有一部分工作是对可见面问题进行形式化的表述，并分析其运算复杂度[GILO78；FOUR88；FIUM89]。例如，Fiume[FIUM89]证明了对象精度可见面算法有一个下界，该下界比排序算法要差。

一般而言，对可见面算法进行比较是很困难的，因为算法之间计算的信息和精度都不同。例如，我们讨论过的算法对对象的种类、对象之间的关系，甚至投影的类别都有限制。如我们将在第14章中看到的，可见面算法的选择也会受到明暗处理类型的影响。如果明暗处理的计算过程很复杂，最好选择一种只计算对象的可见部分明暗度的可见面算法，如扫描线算法等。此时，深度排序算法就不是很好的选择，因为它对所有的对象都做整体的绘制。当交互性能很重要时，常常选择硬件的z缓存算法。BSP树算法对静态场景能够很快地生成新的视图，但当场景发生改变时，需要附加的处理。扫描线算法能够达到很高的分辨率，因为在数据结构中需要详细记录所有影响扫描线的图元。总的来看，实现算法所需的时间以及算法可扩充的程度（如适用于新的图元）都是很重要的因素。

使用可见面算法的一个重要的考虑是是否有硬件支持。如果采用并行的机器，有一点必须引起注意，那就是如果算法利用相关性，则在每个位置都依赖于前一次计算生成的值。利用并发性可能必须放弃某些形式的相关性。光线跟踪非常适合于采用并发的形式执行，它的每个像素都单独进行计算。

习题

13.1 证明：13.1.2节中的变换M保持(a)直线、(b)平面和(c)深度关系。

13.2 给定平面$Ax + By + Cz + D = 0$，用13.1.2节中的矩阵M做变换，求出新的平面方程的系数。

474 13.3 如何对扫描线算法进行扩充以处理具有共享边的多边形？一条共享边是只保存一次（标记为共享边）还是在每一个归属多边形中各保存一次，不做任何标记？在公共边处计算两个多边形的深度时，深度是相等的。如果扫描正进入两个多边形，哪一个多边形是可见的呢？

13.4 考虑z缓存算法、深度排序算法、Warnock算法和BSP树算法，解释穿透多边形都是如何处理的。是作为特殊情况专门处理，还是由基本算法解决？

13.5 习题13.4中提到的算法经过怎样的改进才能够适用于有空洞的多边形？

13.6 z缓存算法的优点之一是对图元的顺序没有要求。但这是否意味着按不同顺序生成的两幅图像的z缓存和帧缓存具有相同的值呢？说明理由。

13.7 考虑两幅同样大小的图像，它们分别用帧缓存和z缓存表示，现要将它们合并在一起。如果已知每幅图像的z_{min}和z_{max}，且原始对应的z值也已知，是否能够将它们正确地合并？还需要其他附加信息吗？

13.8 13.2节提到用整数的z缓存绘制透视投影时产生的z压缩问题。请选择一个透视观察规范和少量的对象点。演示在透视变换过程中，接近投影中心的两个点如何映射到不同的z值，而同样距离但远离投影中心的两点却映射到一个z值。

13.9 a. 假定视见体V在距离F、B处（均为正）分别有前、后裁剪平面，该距离是从VRP开始、沿DOP度量。假定沿DOP度量，从到COP到VRP的距离是w。而且假定前裁剪平面在VRP与COP之间，VRP在COP与后裁剪平面之间（如图6-16所示）。定义$f = w - F$及$b = w + B$，则f是COP到前裁剪平面之间的距离，而b是COP到后裁剪平面之间的距离。现在再类似地假定另一视见体V'并定义f'及b'。将两个视见体规格化后，V的后裁剪平面为$z = -1$，前裁剪平面为$z = A$。对于V'而言，前裁剪平面为$z = A'$。试证明：若$f/b = f'/b'$，则$A = A'$。反之亦然。简言之，在变换到规格化视见体后的z范围，只依赖于从COP到前裁剪平面之间的比值。

b. 由(a)部分可知，在考虑透视效果时，只需要考虑后平面与前平面距离（从COP开始）的比。因此可以简单地研究具有不同前平面距离值的规范视见体。假设有一个规范的透视投影视见体，前裁剪平面为$z=A$，后裁剪平面为$z = -1$，对它做透视变换，得到
475 $z = 0$和$z = -1$之间的平行视见体。请给出用原始z坐标表达变换后新z坐标的公式。（当然，你的答案依赖于A。）假设变换到平行视见体中的z值乘以2^n，然后取整（即映射到整型的z缓存）。找出两个尽可能远的z值，使其在该变换下，映射到相同的整数。（你的答案将依赖于n和A。）

c. 假设图像的比率f/b为R且要求两对象之间在z方向上的距离大于Q时，必须映射到z缓存中的不同值。根据(b)中的工作，试写-算式求出z缓存需要的位数。

13.10 执行光线跟踪时，常常只需要计算光线是否与某一范围相交，而不用求实际的相交点。完成二次方程的光线与球的求交方程（式(13-16)），并说明如何将其简化成仅判断光线与球是否相交。

13.11 通过数值积分，光线跟踪可用于计算对象的质量属性。光线与对象相交的合集给出了光线位于对象内的部分。试通过发射平行光线组来估计对象的体积。

13.12 推导光线与二次曲面的交。修改用于推导光线与球相交的式(13-12)至式(13-15)来处理9.4节中给出的二次曲面的定义。

13.13 实现本章的一个多边形可见面算法，如z缓存算法或扫描线算法。

13.14 实现一个对球与多边形的简单光线跟踪算法，包含自适应超采样。（选择14.1节中的一
476 个光照模型。）采用空间划分或建立包围体的层次结构来改进你的算法的性能。

第14章 光照和明暗处理

这一章我们讨论如何根据物体表面的位置、方向和特性以及照射光源的特性为物体表面加上光照效果。我们给出各种不同的**光照模型**（illumination model），这些光照模型表达确定物体表面给定点处颜色的各种要素。光照模型通常也叫作**光照明模型**（lighting model）或者**明暗处理模型**（shading model）。不过这里我们把明暗处理模型这一术语用于适用范围更广的光照模型。明暗处理模型决定何时运用光照模型以及接受何种参数。比如，某些明暗处理模型为图像中的每个像素调用同一个光照模型，而另外一些只为图像中的某些像素调用某种光照模型，而其他像素的颜色值则通过插值计算得到。

与前一章进行可见面计算所用精度比较，我们将严格区分使用真实物体几何的算法和使用多边形近似的算法，严格区分对象精度的和图像精度的算法，区分那些以单点采样图像精度算法和用更好的过滤器的算法。但在任何情况下，决定某物体在某一像素处是否可见的惟一标准是看通过此像素的投影线上该物体和观察者之间是否存在其他物体。与之相反，光和表面的相互作用却更复杂。或许是为了简化计算或许是因为在图形学领域中不知道更精确的模型，图形学的研究者通常对光和热辐射基本规则采用近似方法。所以，计算机图形学领域中用到的许多光照模型和明暗处理模型是通过不断地拼凑、尝试和简化得到的，虽然这些模型在理论上缺乏坚实基础但在实践中却是可行的。这一章的第一部分涵盖了这样一些简单模型。这些简单模型仍被普遍使用，因为它们能以最小的计算得到引人注目的有用结果。 477

14.1节首先讨论那些仅考虑物体表面单独一点和直接照射它的光源的简单光照模型，首先为单色的表面和光源提出一些光照模型，然后讨论如何将这些计算推广到处理第11章讨论过的颜色系统中。14.2节将介绍一些最常用的使用这些光照模型的明暗处理模型。在14.3节，我们对这些模型进行推广，以模拟表面纹理。

模拟折射、反射和阴影需要一些类似于隐藏面消除并且往往与之结合的额外计算。实际上，这些效果的产生是因为一些被隐藏的表面并不是真的被完全隐藏。它们可以被透视、被反射或者在加上明暗色调的物体表面投下阴影。14.4节和14.5节讨论如何模拟这些效果。

14.6节到14.8节描述那些试图考虑所有表面之间光线交换的全局光照模型：递归光线跟踪和辐射度方法。递归光线跟踪技术将前一章介绍的可见面光线跟踪算法进一步扩展，以确定各像素的可见性、光照以及明暗处理。辐射度方法模拟一个曲面系统的能量平衡，它们在进行当前视点可见性判定计算前就决定了环境中一系列采样点的光照，并且采样点的光照与视点无关。这里所有的光照模型和明暗处理模型都可以在文献[GLAS89；HALL89]中找到。

最后，在14.9节，结合本章和前几章所讨论的光栅化技术，考察了几种不同图形绘制流水线技术，并探讨实现这些技术的途径以建立高效的可扩展的系统。

14.1 光照模型

14.1.1 环境光

可能最简单的光照模型是那个隐含运用在本书前几章的模型，在其中每个物体用一种内蕴亮度进行显示。我们可以想像为这样一个模型，它没有任何外部光源，没有任何反射，物体自身发光，这样的世界当然是不真实的。除非是当该物体创建时赋予物体的不同部分（如一个多面体的

各个面）以不同的色调，否则每个物体将显示为一个单色的轮廓。彩图27展示了这样一种效果。

一个光照模型可以表示为一些变量的**光照方程**，这些变量与带消隐的物体上的点相关。表示这个简单光照模型的光照方程为

478

$$I = k_{\mathrm{i}} \tag{14-1}$$

其中I是所得亮度，系数k_{i}是该物体内设光亮度。因为此光照方程中不包含任何依赖于受照射的点的位置的项，所以对整个物体只需要计算一次。对物体上一个或更多点对光照方程求值的过程通常称为**照明**物体。

现在想像一下，物体不是自发光的，而是存在一个漫射的无方向的光源，环境中存在的光是经过多个表面多次反射所得到的结果，这种光通常被称为**环境光**（泛光）。如果我们假设环境光从各个方向均衡地照射到所有物体表面，那么光照方程为

$$I = I_{\mathrm{a}}k_{\mathrm{a}} \tag{14-2}$$

其中I_{a}是环境光亮度，假设对所有物体都是常数。从物体表面所反射的环境光的量由**环境反射系数**k_{a}决定，它的取值范围为0到1。环境光反射系数是一个**材质属性**。与我们将要讨论的其他各种材质属性一样，它可以看成是刻画构造该物体表面所用材质的特性。像其他的一些性质一样，环境光反射系数是一个适当的经验值，并不直接对应于真实材料的任何物理属性。而且对环境光本身影响不是很大。正如我们将看到的，它通常被用于考虑所有在实际中光能够到达物体但却不能为光照方程所计算的复杂情况。彩图27给出了环境光下的光照效果。

14.1.2 漫反射

虽然在环境光照射下物体的亮度差不多与环境光的亮度成正比，但物体表面仍呈现出单一的光照效果。现在考虑以**点光源**照射的一个物体，该点光源的光线从一点向四周均匀发散。根据到光源的不同方向和距离，物体从一部分到另一部分的辉度将有所不同。

1. 朗伯反射

暗的粗糙表面，如粉笔，展现出**漫反射**效果，也被称为**朗伯反射**。因为这些表面从各个方向等强度地反射光，因而从各个视角物体表面呈现出相同的亮度。对于给定的表面，该表面亮度仅依赖于图14-1中该表面到光源的方向 $\overline{L}$ 与物体表面法向 $\overline{N}$ 的夹角。让我们看看为什么会发生这种情况。在这里有两个因素起作用。第一，图14-2显示照射到物体表面的一束光线覆盖了该物体表面一定的面积，该面积大小与该光束与物体表面法向 $\overline{N}$ 所成的夹角θ的余弦成反比。如果光束有一极微小的截面微面元$\mathrm{d}A$，那么该光束在物体表面上的截面积为$\mathrm{d}A/\cos\theta$。因此，对一入射光束，落在$\mathrm{d}A$面积上的光的能量与$\cos\theta$成比例。这对所有表面都成立，而与构造该表面的材质无关。

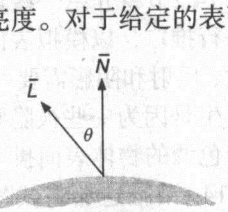


图14-1 漫反射

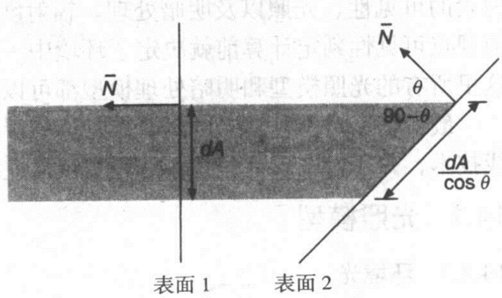


图14-2 具有极微小横截面面积 $\mathrm{d}A$的以入射角θ入射的光束（用二维截面图表示）照射到$\mathrm{d}A/\cos\theta$的面积

第二，我们必须考虑观察者所看到的光的量。朗伯表面具有这样一个性质，通常被称为朗伯定律，即从单位微面元$\mathrm{d}A$向观察者反射的光的量直接与物体表面到观察者的方向与物体表面法向 $\overline{N}$ 夹角的余弦成正比。但是所看到的物体

表面面积量与这个夹角的余弦成反比，因此这两个因子相抵消。比如，当视角增大，观察者看到更大的物体表面面积，但从此方向上单位面积表面所反射的光量也相应地成比例地少了。因此，对朗伯表面，被观察者所看到的光量是独立于视线方向并且仅与入射光角θ的余弦$\cos\theta$成比例的。

漫反射光照方程是

$$I = I_p k_d \cos\theta \tag{14-3}$$

I_p是点光源的亮度。材质的**漫反射系数**k_d是一个从0到1之间的常数并随材质的不同而变化。如果光源对它所照射的点有任何直接作用，则该光源入射角θ必须在0°到90°之间。这意味着该表面可以进行**自遮挡**处理，因此从表面一点的后面所投射的光并不照射到它。在这里和下面的等式中，与其不显式地包含一个$\max(\cos\theta,0)$项，还不如假设θ在有效范围内。当我们希望为一个自遮挡的物体表面加上光照时，我们可以用$\text{abs}(\cos\theta)$来倒转它们的表面法向。这使得该表面的两面得以被以同样的方式处理，好像该表面被两个相对的光源照射。

假设向量$\overline{N}$和$\overline{L}$已经被规格化（参见5.1节）。我们可以用点积重写式(14-3)

$$I = I_p k_d (\overline{N} \cdot \overline{L}) \tag{14-4}$$

表面法向$\overline{N}$可以用第9章所讨论的方法计算。如果多边形的法向预先计算并以对多边形顶点做变换的同一矩阵进行变换，那么重要的是不做诸如错切或不同的缩放那样的非刚体模型变换，因为这些变换不是保角变换，而且可能使得一些法向量不再垂直于它们的多边形。5.7节给出了对物体进行任意变换时，对法向做转换的正确方法。在任何情况下，光照方程必须在世界坐标系下（或者与之尺寸相同的任何坐标系）计算，因为规格化和透视投影变换都会改变θ值。 479~480

如果一个点光源与它所照射的物体足够远，这使得它与所有法向相同的表面成同样的入射角。这种情况下，这种光称为*方向光源*，此时$\overline{L}$是常数。

图14-3展示了一个点光源照射下的球体的一系列图片。该明暗处理模型用式(14-4)的光照模型为该球体每一个可见的像素计算亮度。以此方式加上明暗的物体看起来很粗糙，就像一支手电在一个相当暗的房间照射一个物体。所以，通常加入环境光项以产生更真实的光照方程

$$I = I_a k_a + I_p k_d (\overline{N} \cdot \overline{L}) \tag{14-5}$$

图14-4是用式(14-5)产生的画面。

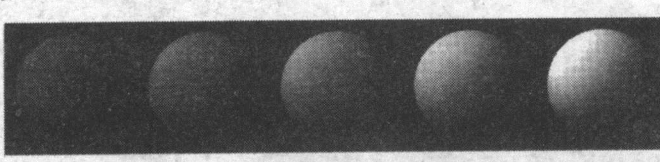


图14-3　用漫反射模型（式(14-4)）加上明暗的球。从左至右，$k_d = 0.4$，0.55，0.7，0.85，1.0。（由哥伦比亚大学David Kurlander提供。）

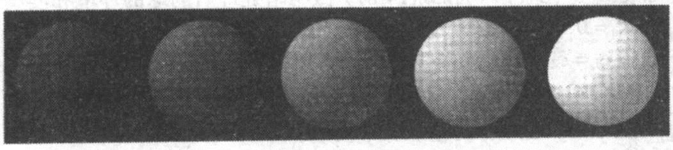


图14-4　用环境和漫反射（式(14-5)）加上明暗的球。对所有球，$I_a = I_p = 1.0$，$k_d = 0.4$，$k_a = 0.0$，0.15，0.30，0.45，0.60。（由哥伦比亚大学David Kurlander提供。） 481

2. 光源的衰减

由视点处照射，如果两个具有同样材质的平行平面的投影在图像中相互重叠，那么无论它

们与光源的距离如何不同，由式(14-5)生成的画面将无法区别光线是从哪个表面照射到哪个表面的。为处理这种情况，我们引进了光源衰减因子f_{att}，从而有

$$I = I_a k_a + f_{att} I_p k_d (\overline{N} \cdot \overline{L}) \tag{14-6}$$

f_{att}的选择显然应考虑到这样一个事实，即从点光源到达物体表面某部分的光能量是以d_L的平方的倒数衰减的，d_L是点光源到物体表面的距离。在这种情况下，

$$f_{att} = \frac{1}{d_L^2} \tag{14-7}$$

然而，实际上这种假设并不总是对的。如果光很远，$1/d_L^2$的变化并不大；如果它很近，则变化很大，这样，$\overline{N}$与$\overline{L}$之间相同的夹角θ却因在不同表面而产生相当不同的明暗效果。虽然这种运行情况对点光源是正确的，但实际中我们所见的物体通常并不为点光源所照射，也不具有计算机图形学中简化了的光照模型计算为物体加上的明暗阴影效果。为使效果更逼真，早期的图形学研究者通常使用一个放在视点处的点光源。他们期望用f_{att}来近似模拟在观察者和物体之间的大气衰减现象（见14.1.3节），同时近似模拟从光源到物体的能量密度的衰减。一种容许比简单的平方规律衰减效果更好的可行折衷是

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right) \tag{14-8}$$

这里c_1、c_2和c_3是用户定义的与光源相关的常数。当光源很近时，常数c_1防止分母变得太小，同时该表达式被限定在最大值1以内以确保其总是衰减的。图14-5即是采用这个常数取不同值的光照模型所显示的一系列不同的效果。

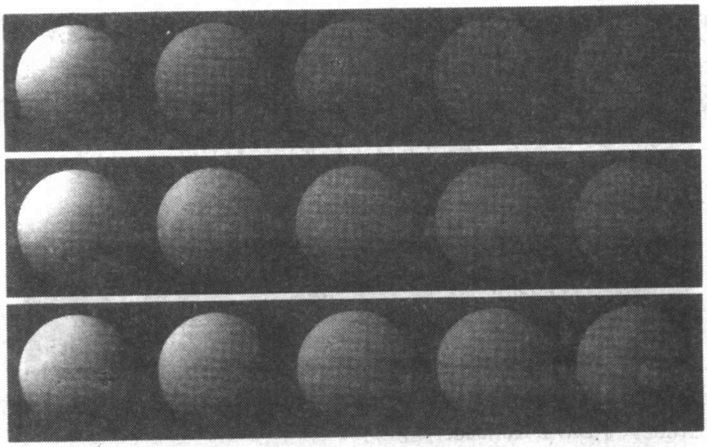


图14-5 用带点光源衰减项（式(14-6)和式(14-8)）做明暗处理后的球的环境和漫反射。对每个球，$I_a = I_p = 1.0$，$k_a = 0.1$，$k_d = 0.9$。从左至右，从光源到球的距离为1.0, 1.375, 1.75, 2,125和2.5。第一行，$c_1 = c_2 = 0.0$，$c_3 = 1.0(1/d_L^2)$；第二行，$c_1 = c_2 = 0.25$，$c_3 = 0.5$；第三行，$c_1 = 0.0$，$c_2 = 1.0$，$c_3 = 0.0(1/d_L)$。（由哥伦比亚大学David Kurlander提供。）

3. *彩色光和表面*

至今我们描述的都是单色光和表面。彩色光和表面通常通过为颜色模型中每个组成部分写不同的方程的方式进行处理。我们通过O_d的一个值来表示一个物体的**漫射颜色**的每一部分。比如，在RGB颜色系统中，三元组（O_{dR}, O_{dG}, O_{dB}）定义了一个物体的漫射红、绿、蓝组成部分。在这种情况下，照射光的三个主要组成部分I_{pR}, I_{pG}和I_{pB}分别以$k_d O_{dR}$, $k_d O_{dG}$ 和$k_d O_{dB}$的比例反射。

所以，对红色部分来说，

$$I_R = I_{aR}k_aO_{dR} + f_{att}I_{pR}k_dO_{dR}(\overline{N} \cdot \overline{L}) \tag{14-9}$$

对绿色和蓝色部分I_G和I_B采用相似的等式。使用单一的系数来放大或缩小等式中的表达式使用户不需改变它们的分量的比例就可以控制环境反射或漫反射的量。另一个表达更紧凑但不方便控制的公式则简单地为每个分量采用不同的常数，比如，用k_{aR}代替k_aO_{dR}，用k_{dR}代替k_dO_{dR}。

这里做了一个简化的假设，即三色模型能完全模拟光与物体的相互作用。这种假设是错误的，但它更容易实现且常常产生可接受的图片。在理论上，应该在模拟的光谱范围对光照方程进行连续的取值；但在实际中，只在一些间断光谱样本上进行取值。不是将我们自己约束于特定的颜色模型，我们在光照方程中用下标λ显式地标明那些与波长相关的量。这样，式(14-9)变为

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}k_dO_{d\lambda}(\overline{N} \cdot \overline{L}) \tag{14-10}$$

14.1.3 大气衰减

为了模拟从物体到观察者的大气衰减，许多系统提供**深度提示**（depth cueing）。在这种最 482~483
初起源于向量图形硬件的技术中，用比近的物体低的亮度绘制更远的物体。PHIGSPLUS标准推荐了一种深度提示方法，这种方法使得由于居间大气层所引起的颜色变化的近似计算成为可能。在NPC中分别定义前后深度提示参考平面，每个平面分别与一个取值为0到1的缩放因子s_f、s_b相联系。这个缩放因子决定了原始亮度与深度提示颜色的亮度混合。目标是为了修正前面计算的I_λ值以产生最终显示的深度提示值I'_λ。给定物体的z坐标z_o，可以推得一个缩放因子s_o，并将其用于I_λ与$I_{dc\lambda}$之间的插值中以决定

$$I'_\lambda = s_oI_\lambda + (1 - s_o)I_{dc\lambda} \tag{14-11}$$

如果z_o在深度提示前参考平面的z坐标值z_f前，那么$s_o = s_f$；如果z_o在深度提示后参考平面的z坐标值z_b后，那么$s_o = s_b$。最后，如果z_o在两参考平面之间，那么

$$s_o = s_b + \frac{(z_o - z_b)(s_f - s_b)}{z_f - z_b} \tag{14-12}$$

s_o与z_o的关系如图14-6所示。图14-7表示加上深度提示的明暗效果的不同球。为避免使等式复杂化，在进一步提出光照模型时，我们忽略深度提示效果。

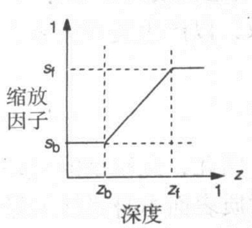


图14-6 计算大气衰减缩放因子

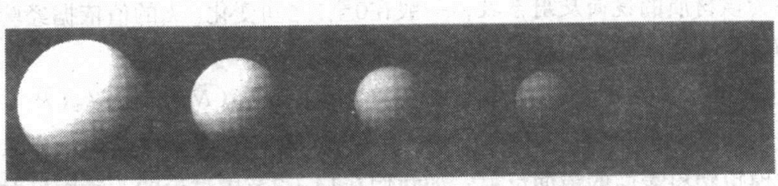


图14-7 用深度提示（式(14-5)，式(14-11)和式(14-12)）加上明暗的球。到光源的距离是常值。对每个球，$I_a = I_p = 1.0$，$k_a = 0.1$，$k_d = 0.9$，$z_f = 1.0$，$z_b = 0.0$，$s_f = 1.0$，$s_b = 0.1$，半经 = 0.09。从左至右，球的最前面一点z值为1.0, 0.77, 0.55, 0.32和0.09。（由哥伦比亚大学David Kurlander提供。）

14.1.4 镜面反射

在任何发光表面上可以观察到**镜面反射**现象。用明亮白光照射一个苹果：强光由镜面反射引起，而从其他部分反射的光则是漫反射的结果。同时可注意到，在强光处苹果并非呈现出红色，而是入射光的颜色——白色。像蜡制的苹果或发光的塑料这样的物体，都有一个透光的表

面；比如，塑料是典型的由沉淀于透明材料中的颜色颗粒所构成的。从无色的表面镜面反射的
484 光通常呈现出与光源相同的颜色。

现在移动您的头部，可观察到强光也在移动。出现这种现象是因为闪光表面在不同方向上反射光不相同；在一个理想的闪光表面，如一个完全平坦的镜子，光只在反射方向 $\overline{R}$ 上被反射，该反射方向与 $\overline{L}$ 是关于 $\overline{N}$ 对称的。因此，观察者只有在图14-8中的角 α 为0时，才能观察从镜子反射的镜面光。这里 α 是 $\overline{R}$ 到视点 $\overline{V}$ 的方向的夹角。

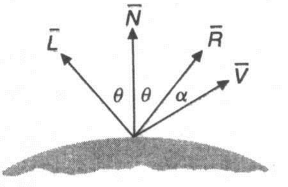


图14-8 镜面反射

1. Phong光照模型

Phong Bui-Tuong[BUIT75]为不完全平坦的反射面（如苹果）提出了一种非常受欢迎的光照模型。它假设当 α 为0时，发生的镜面反射最大，并且随着 α 的增大镜面反射大幅度地减弱。这种快速衰减效果由 $\cos^n\alpha$ 来近似，这里 n 是这种材质的**镜面反射指数**。依据所模拟的表面材质，n 的值从1到几百变化不等。当 n 为1时，强光区域范围较大、衰减也较缓慢，而 n 值较大时则模拟强烈的聚焦强光（图14-9）。对一个完美的反射体，n 是一个无穷大的值。与以前一样，我们把 $\cos\alpha$ 的负值当作0处理。Phong光照模型基于Warnock等研究者早期的工作[WARN69]，他们曾用项 $\cos^n\theta$ 来模拟放置在视点处的光源的镜面反射效果。然而Phong是第一个考虑观察者与光源可以放置在任意位置的人。

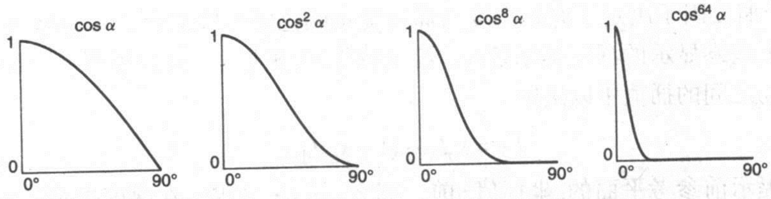


图14-9 用在Phong光照模型中的不同 $\cos^n\alpha$ 值

镜面反射的入射光的量依赖于入射角 θ。如果 $W(\theta)$ 是镜面反射光的比例，那么Phong模型是：

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}\,[k_dO_{d\lambda}\cos\theta + W(\theta)\cos^n\alpha] \tag{14-13}$$

如果反射方向 $\overline{R}$ 和视点方向 $\overline{V}$ 已规格化，那么 $\cos\alpha = \overline{R}\cdot\overline{V}$。此外，$W(\theta)$ 通常设为常数 k_s，k_s 称为该材质的**镜面反射系数**，一般在0至1之间变化。k_s 的值依据经验选取，以产生美观的令人愉悦的效果。那么，式(14-13)可以重写为

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}) + k_s(\overline{R}\cdot\overline{V})^n] \tag{14-14}$$

485 注意，Phong光照模型中的镜面反射分量的颜色不依赖于任何材质属性。因此，此模型能很好地模拟塑料表面的镜面反射。如我们在14.1.7节中讨论的，镜面反射受材质表面本身属性的影响，并且，当该材质表面是不同材料组合而成时，镜面反射通常具有不同于漫反射的颜色。我们可以通过修改式(14-14)中的一个初步近似值来提供这种效果。

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}) + k_sO_{s\lambda}(\overline{R}\cdot\overline{V})^n] \tag{14-15}$$

这里 $O_{s\lambda}$ 是物体的**镜面反射颜色**。图14-10给出用式(14-14)中不同 k_s 和 n 值所计算的球面的光照。

2. 计算反射向量

计算 $\overline{R}$ 要求 $\overline{L}$ 关于 $\overline{N}$ 的对称向量。如图14-11所示，这可以通过简单的几何计算完成。因为 $\overline{N}$ 和 $\overline{L}$ 已规格化，$\overline{L}$ 到 $\overline{N}$ 上的投影是 $\overline{N}\cos\theta$。注意，这里 $\overline{R} = \overline{N}\cos\theta + \overline{S}$，$|\overline{S}|$ 为 $\sin\theta$。但

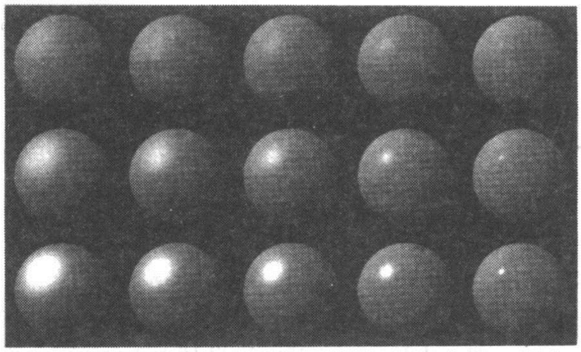


图14-10 使用Phong光照模型和不同的k_s和n值加上明暗的球面。对每个球，$I_a = I_p = 1.0$，$k_a = 0.1$，$k_d = 0.45$。从左至右，$n = 3.0$，5.0，10.0，27.0，200.0。从上到下，$k_s = 0.1$，0.25，0.5。（由哥伦比亚大学David Kurlander提供。）

通过向量减法和全等三角形，$\overline{S} = \overline{N}\cos\theta - \overline{L}$。所以$\overline{R} = 2\overline{N}\cos\theta - \overline{L}$。用$\overline{N} \cdot \overline{L}$代替$\cos\theta$，$\overline{R} \cdot \overline{V}$代替$\cos\alpha$，得到

$$\overline{R} = 2\overline{N}(\overline{N} \cdot \overline{L}) - \overline{L} \tag{14-16}$$

$$\overline{R} \cdot \overline{V} = (2\overline{N}(\overline{N} \cdot \overline{L}) - \overline{L}) \cdot \overline{V} \tag{14-17}$$

如果光源在无穷远处，对所有给定的多边形，$\overline{N} \cdot \overline{L}$都是常数，而$\overline{R} \cdot \overline{V}$则在给定的多边形上要变化。对曲面表面或者光源不在无限远处的，$\overline{N} \cdot \overline{L}$和$\overline{R} \cdot \overline{V}$在表面上都要变化。

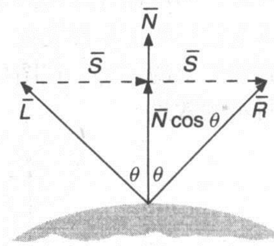


图14-11 计算反射向量

3. 中间向量

Phong光照模型的另一个公式使用**中间向量**$\overline{H}$，之所以这样叫是因为它的方向正如图14-12中所示处在光源方向和观察者方向中间。$\overline{H}$也是人们所知的为最强强光方向。如果调整表面位置使得它的法向与$\overline{H}$相同，则观察者将会看到最明亮的镜面强光，因为$\overline{R}$和$\overline{V}$会指向同一个方向。新的镜面反射项可以表示为$(\overline{N} \cdot \overline{H})^n$，其中$\overline{H} = (\overline{L}+\overline{V})/|\overline{L}+\overline{V}|$。当光源和观察者都在无穷远处时，$\overline{N} \cdot \overline{H}$的使用提供了计算优势，因为$\overline{H}$是常数。注意，$\overline{N}$与$\overline{H}$的夹角$\beta$并不等于$\overline{R}$与$\overline{V}$的夹角$\alpha$，因此，同样的强光指数$n$在这两个公式中将产生不同的结果（见习题14.1）。虽然用项$\cos^n$允许生成明显的光滑表面，但应该记住该项基于的是经验观察结果，而非镜面反射过程的理论模型。

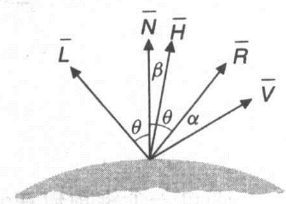


图14-12 中间向量$\overline{H}$在光源方向和观察者方向的中间

14.1.5 点光源模型的改进

真实的光源并不在各个方向等同地发散光能。Warn[WARN83]提出一些易于实现的光照控制方法，也就是说，可以将这些控制加入到任何光照方程中，用来模拟摄影者使用的光的定向方法。在Phong模型中，点光源只有一个亮度值和一个位置，而在Warn的模型中，通过一个假定的镜面反射表面上的点来模拟一个光源L（如图14-13所示）。该表面为点光源L'从方向$\overline{L}'$照射。假设$\overline{L}'$垂直于这个假设的反射平面，那么，我们可以用Phong光照模型以$\overline{L}$与$\overline{L}'$之间的夹角γ决定该表面任一点所模拟的光的亮度。如果进一步假设反射面仅反射镜面光并且镜面反射系数为1，那么在表面上一点处光的亮度为

$$I_{L'\lambda} \cos^p \gamma \tag{14-18}$$

其中，$I_{L'\lambda}$是假设的点光源的亮度；p是反射物的镜面反射指数；γ是$-\bar{L}$与假设表面法向$\bar{L}'$的夹角，它是到L'的方向。式(14-18)模拟了一个对称的有向光源，其对称轴为$\bar{L}'$，是所模拟的光的假设照射方向。用点积我们可以将式(14-18)写为

$$I_{L'\lambda}(-\bar{L}\cdot\bar{L}')^p \tag{14-19}$$

另外，我们将负的点积视为0。因此，可用式(14-19)代替式(14-15)或其他任何光照方程中的光源亮度项$I_{p\lambda}$。p值越大，则光越集中在$\bar{L}'$方向上。因此，大的p值可以模拟高度方向性的点光源，而小的值则模拟一个均匀发散的柱光源。如果p为0，那么光就像一个均匀发散的点光源。图14-15a至图14-15c展示了不同p值的效果。

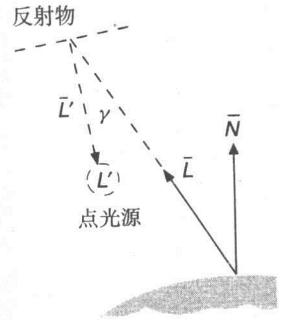


图14-13 Warn的光照模型。通过由点光源照射下的一点的镜面反射来模拟一个光源

为了将光的效应局限于场景中的有限区域内，Warn实现了**挡板**（flap）和**锥体**（cone）的模拟。挡板仿照专业摄影中的挡光板制作的，将光的效果限制于x、y、z坐标中的预定范围。每个光有六个挡板，分别对应于用户定义的各坐标轴的最小值和最大值。在决定一个点的色深时，只有当该点的坐标值在那些开着的挡板所定义的最小和最大坐标值的范围内时，才对一个光源进行光照模型的计算。比如，如果$\bar{L}'$平行于y轴，那么，就像摄影灯中的挡光板一样，x轴、z轴上的挡板可以严格限制此光的效果。图14-14a描述在这种情形下x轴上的挡板的运用。y轴上的挡板也可以在这里用于限制光线，但这种方式在实际中并不存在，它只容许在一定光源范围内的物体受到该光源的照射。在图14-15d中，立方体与坐标系统对齐，因此两对挡板就可以产生如图14-15所示的效果。

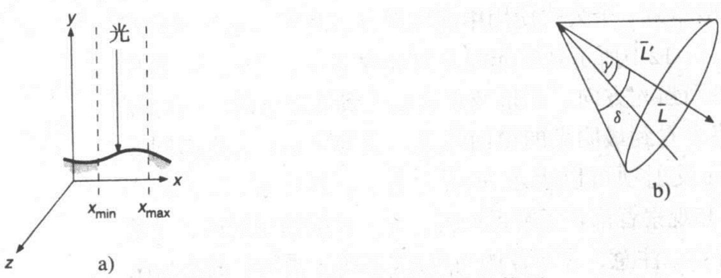


图14-14 a) 挡板和b) 锥体的使用

通过一个锥顶放在光源处而轴落在$\bar{L}'$上的锥体，Warn产生了轮廓分明的聚光效果。如图14-14b所示，通过计算仅当$\gamma<\delta$时（或者当$\cos\gamma>\cos\delta$时，因为已经计算了$\cos\gamma$）的光照模型，一个具有δ大小生成角的锥体可以用来限制光源的影响范围。PHIGSPLUS中的光照模型包含了Warn的$\cos^p\gamma$项和锥角δ。图14-15e显示了如何用锥体来限制图14-15c光源。彩图21是一幅用Warn的光照控制绘制的汽车。

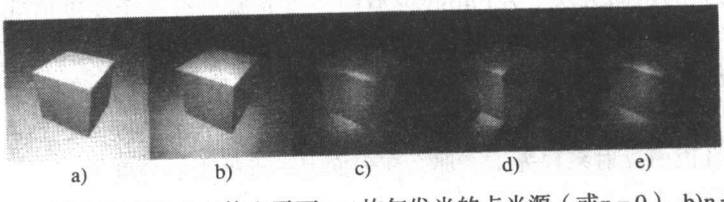


图14-15 用Warn的光源控制照射的立方体和平面。a)均匀发光的点光源（或$p=0$），b)$p=4$，c)$p=32$，d)挡板的效果，e)$\delta=18°$的锥体。（由哥伦比亚大学David Kurlander提供。）

14.1.6 多光源

如果有m个光源，那么每个光源项的和是

$$I_\lambda = I_{a\lambda}k_a O_{d\lambda} + \sum_{1\leq i\leq m} f_{\text{att}_i} I_{p\lambda_i}[k_d O_{d\lambda}(\overline{N}\cdot\overline{L}_i) + k_s O_{s\lambda}(\overline{R}_i\cdot\overline{V})^n] \tag{14-20}$$ 486~488

这个和式可能隐含了一个新的错误，即I_λ可能超出了最大的可显示像素值。（虽然对一个光源这个错误也可能发生，但我们可以通过选择适当的f_{att}和材料参数很容易地避免它。）有一些方法可以用来避免溢出。最简单的方法是将每个值取上限。另一个方法是一起考虑所有像素的I_λ值。如果至少有一个太大，那么将每个值除以这个最大值以保证色彩和饱和度。如果在显示前可以计算出所有像素的值，那么可以将图像处理中的变换作用在整幅图上以使所有值在所要求的范围内。Hall[HALL89]对这些技术和另外一些技术进行了比较。

14.1.7 基于物理的光照模型

在此之前所讨论的光照模型大部分基于我们的常识，是图形学中的一些实际的方法。虽然，所用等式近似地描述了光线如何与物体相互作用，但是它们并没有一个物理基础。在本节里，我们将讨论一些基于物理的光照模型，其中一部分内容将基于Cook 和Torrance的工作[COOK82]。

到目前为止我们一直都在使用*亮度*（intensity）这个词，但却没有严格定义，并经常用它来非正式地描述一个光源的亮度、一个面上的一点甚至是一个像素的亮度。现在用热辐射学中的辐射计量术语来正式解释我们使用的术语。热辐射学是我们理解光线如何与物体相互作用的基础[NICO77；SPARR78；SIEG81；IES87]。首先，我们介绍**光通量**（flux），即单位时间内传播的光能，单位为瓦特。为了计算在一个方向上接收或发送的光通量的值，我们还需要引入**立体角**的概念，即圆锥体的锥顶角。立体角可以以锥顶在球心上的圆锥所截取球的球面面积来计量。**单位立体角**（sr）即为截取的球面积等于该球半径r平方的圆锥的立方体的立体角。如果球面上有一点，我们关心的是位于该点的半球。因为一个球面面积为$4\pi r^2$，则对于一个半球，有$4\pi r^2 / 2r^2 = 2\pi$个sr。假设以物体表面上一点为投影中心，将一个物体投影到以该点为球心的球面上，该物体所张的立体角ω便是用球面上该物体投影面积除以半球半径的平方，（这个除法消除了立体角对球面大小的依赖。）

辐射强度（radiant intensity）定义为一特定方向一个单位立体角内传播的光通量，以W / sr计量，当我们用亮度表示一个点光源时，通常是指它的辐射强度。

辐射光亮度（radiance）是每一个单位透视缩小曲面面积上的辐射强度，用W / (sr · m^2)来度量。**透视缩小**（foreshorten）**曲面面积也叫投影面面积**，指的是该表面到垂直于辐射方向的平面上的投影。透视缩小曲面面积可以通过表面面积乘以$\cos\theta_r$求得，其中θ_r为辐射光线相对于表面法线的角。一个小的立体角$d\omega$可以通过物体的透视缩小曲面面积除以立体角所在顶点到物体的距离的平方来近似。489 以前当我们说到一个表面的*亮度*时，通常就是指它的辐射光亮度。**辐照度**（irradiance），通常又称为**光通量密度**，是单位非透视缩小曲面面积上所入射的光通量，用W/m^{2}来度量。

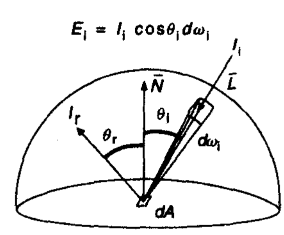


图14-16 反射光的辐射光亮度和入射光的辐照度

在图形学中，我们关心表面入射光与该表面反射光和透射光的关系。考虑图14-16，入射光的辐照度为

$$E_i = I_i(\overline{N}\cdot\overline{L})\,d\omega_i$$

其中I_i为入射光的辐射光亮度，$\overline{N}\cdot\overline{L}$为$\cos\theta_i$。因为辐照度也是以单位面积表示的，而辐射光亮度是以单位透视缩小的曲面面积表示的，所以乘以$\overline{N}\cdot\overline{L}$即将其转换为以单位非透视缩小曲面面积表示。

在求I_r（反射光的辐射光亮度）时仅仅考虑I_i（入射光的辐射光亮度）是不够的，必须同时

考虑E_i（入射光的辐照度）。例如，若两束入射光线有着相同的辐射强度（W/sr），但有着较大立体角的入射光相应地有较大的E_i，进而相应地导致物体看起来更亮一点。我们把某个方向上反射光的辐射光亮度与相应的入射光的辐照度（光通量密度）的比称为**双向反射率**ρ，它是一个与入射光和反射光角度有关的函数

$$\rho = \frac{I_r}{E_i}$$

如我们所见，在计算机图形学中，通常把双向反射率看成是漫反射分量和镜面反射分量的组合。因此有

$$\rho = k_d\rho_d + k_s\rho_s.$$

其中ρ_d和ρ_s分别是双向漫反射率和双向镜面反射率，而k_d和k_s则分别是本章前面引入的漫反射系数和镜面反射系数。

由应用物理学家提出的Torrance-Sparrow表面模型[TORR66；TORR67]是一个关于反射面的物理模型。Blinn首先把Torrance-Sparrow模型应用于计算机图形学，给出详细的数学细节并在［BLIN77a］中将其与Phong模型进行了比较，而Cook和Torrance[COOK82]则在该模型的实现中，首次模拟了反射光的光谱组成。

在Torrance-Sparrow模型中，物体表面被看成一组各向同性的平面微面元，每一个微面元都是非常光滑的反射面。这些**微面元**的几何特征和分布特征以及光线的方向（假设光线从无穷远处发出，因此所有光线都是相互平行的）决定了镜面反射的亮度和方向是I_P（点光源的亮度）、$\overline{N}$、$\overline{L}$和$\overline{V}$的函数。实验测量表明，实际的反射效果和用此模型来预测的反射效果相当一致[TORR67]。

490 Cook和Torrance使用以下公式计算双向反射率中的镜面反射分量：

$$\rho_s = \frac{F_\lambda}{\pi}\frac{DG}{(\overline{N}\cdot\overline{V})(\overline{N}\cdot\overline{L})} \tag{14-21}$$

其中D是微面元方向的分布函数，G是几何衰减因子，它代表各微面元之间遮挡和阴影效果，而F_λ则是用菲涅耳公式计算出来的菲涅耳项（将在后面介绍）。该公式在镜面反射时表示了每个光滑微面元的反射光与入射光的关系。分母中的π用来解释表面的粗糙程度（要想了解该公式是如何推导的请参见[JOY88，pp.227-230]）。$\overline{N}\cdot\overline{V}$项使得公式与观察者在一块按透视缩小曲面的曲面面积内所看到的表面积（当然也是微面元的数目）成比例，而$\overline{N}\cdot\overline{L}$项则使得公式与光在一块按透视缩小曲面面积内所看到的表面积成比例。式(14-21)各组成项的详细内容，请见[FOLE90]的16.7节，这里仅给出结果。

彩图40给出了两个用Cook-Torrance模型绘制的两个铜光瓶，两者都使用了铜的双向反射系数作为漫射项。第一幅图使用了塑料镜面的反射系数来模拟镜面反射项，所示结果相似于用式(14-14)所表示的原始Phong光照模型所产生的结果。第二幅图则采用了铜质镜面的反射系数来模拟镜面反射项。请注意，如何计算镜面反射光颜色对入射角大小和表面材质的依赖以产生更为逼真的金属表面图案画面。

通常，无论对非光导体还是光导体，环境、漫射和镜面分量都是该物质的颜色。混合物体，例如塑料，它的漫射和镜面分量通常有不同的颜色。金属通常没有漫射，但它的镜面光颜色将在该材料颜色和θ_i接近90°时光源的颜色之间变化。基于这种观察，人们提出了一种使用式(14-15)对Cook-Torrance模型的粗略近似方法，其中$\theta_{s\lambda}$的取值取决于对θ_i的一个查找表的插值。

Cook-Torrance光照模型已有几种增强和推广，若干例子可见[KAJI85；CABR87；WOLF90]。最近由He等人[HE92]采用基于物理模型演示了一个快速、精确的方法。注意，[HE92]是一个多媒体出版物，它允许读者交互式探索式(14-21)许多项的效果，十分有趣。

14.2 多边形的明暗处理模型

显然，我们可以通过计算表面上任何可见点的法向并为该点调用所需的光照模型来给任何面加上明暗色调。但是这种最原始的明暗处理模型代价昂贵。在这一节中，我们为由多边形和多边形网格所定义的表面描述更为有效的明暗处理模型。491

14.2.1 恒定明暗处理

最简单多边形光滑明暗处理模型是**恒定明暗处理**，通常也称为**面片明暗处理**或**挡板明暗处理**。该方法用光照模型对整个多边形只计算一次亮度值，用于处理整个多边形的明暗效果。实质上，只是为每个多边形进行一次光照模型的值采样，并在多边形内保持这个值来重建此多边形的明暗色调。在以下假设成立的情况下这种方法是正确的：

1) 光源在无穷远处，因此在多边形范围内$\bar{N} \cdot \bar{L}$是常量。

2) 观察者位于无穷远处，因此在多边形范围内$\bar{N} \cdot \bar{V}$是常量。

3) 多边形代表了所模拟的真实表面，而不是对物体曲面表面的一个近似。

如果使用的是一个输出一系列多边形的可见面判定算法，如列表优先级算法，恒定明暗处理可以利用这个处处为单色的二维多边形图元。

如果前两个假设中的任何一个是错误的，那么，如果我们仍用恒定明暗处理，则需要一些方法为$\bar{L}$和$\bar{V}$分别决定一个值。比如，该值可以是为多边形中心计算的值或者为多边形第一个顶点计算的值。当然，恒定明暗处理不会在多边形上产生本应在此情况下发生的明暗的变化。

14.2.2 插值明暗处理

为减少多边形每点光照方程的计算，Wylie、Romney、Evans和Erdahl[WYLI67]首先提出了**插值明暗处理**多边形光照计算替代方法。在这里，三角形内各处的明暗度信息是通过对其顶点的明暗值的线性插值得到的。Gouraud[GOUR71]将这种技术推广至任意多边形。该技术非常容易用扫描线算法来实现，因为扫描线算法即通过跨度端点处的z值来插值计算得到整个跨度上的z值。

为提高效率，可以使用类似与13.2节中提出的用其决定像素z值的差分方程。虽然对z值进行插值是正确的（假设多边形是平面的），但却必须注意对明暗进行插值处理却不是，因为它仅仅是对多边形上各点处的光照模型的计算的近似。

最后的这个假设（即多边形正确地表示了所模拟的表面）却往往是错误的。它对最终图像的影响比其他两个假设的错误所带来的影响更大。许多物体是曲面的，而不是多边形的。然而，仍用多边形网格表示只是为了便于使用高效的多边形可见面判定算法。下一步我们讨论如何绘制多边形网格以使它尽可能看起来更像曲面。492

14.2.3 多边形网格的明暗处理

假设用多边形网格来近似曲面表面。如果网格中的每个多边形面单独进行明暗处理，那么它就很容易与周围不同方向的面区分开来，从而产生如彩图28中所示的面效果。如果多边形以恒定明暗处理、插值明暗处理甚至单像素光照计算进行绘制，都将产生这个效果，因为相邻的两个具有不同方向的多边形在它们的边界都具有不同的亮度。结果表明，使用更精细的多边形网格这种简单解决方法很低效，因为在两个相邻面上所觉察到的色调的不同被马赫带效应加剧（由马赫于1865年发现并在[RATL72]中详细描述），在具有不连续的亮度大小或变化的边界处，马赫带效应将加剧其亮度变化。在两个面的边界，暗的面看起来更暗，而亮的面看起来更亮。图14-17显示，在两种不同情况下真正的和可察觉到的沿表面的亮度变化。

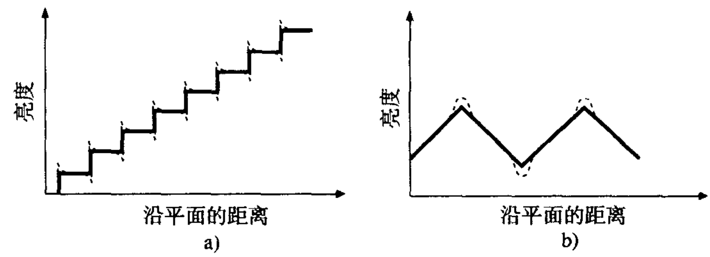


图14-17 真正的和在马赫带效应中察觉到亮度的两个例子。虚线是察觉到的亮度，实线是真实亮度

马赫带效应是由眼中感光细胞的**侧向抑制**所引起的。感光细胞接受的光越多，它对邻接感光细胞的反应则抑制得越大。感光细胞对光反应的抑制与邻接感光细胞的距离成反比。在亮度不连续处较亮一侧的感光细胞中，紧邻该不连续处的将比远离该不连续处的感光细胞具有更强的反应，因为它们受到较暗一侧的邻接感光细胞较少的抑制。同样，在较暗一侧的感光细胞中，处在暗区的将比那些离该暗区远的感光细胞有较弱的反应，因为它们受处在较亮区域相邻感光细胞较多的抑制。彩图28中的马赫带效应非常明显，特别是在那些颜色相近的相邻多边形之间。

我们已描述的多边形明暗处理算法都是单独决定各多边形的明暗色调的。另有两种基本的多边形网格明暗处理模型，这两种明暗处理模型利用多边形的相邻信息来模拟平滑表面。以复
493 杂度（或真实感效果）增加的顺序，通称为Gouraud明暗处理和Phong明暗处理，分别以提出它们的研究者的名字命名。当前三维图形工作站一般通过硬件和固件的结合来支持这两个方法中的一个或两个。

14.2.4 Gouraud 明暗处理技术

Gouraud明暗处理[GOUR71]又被称为**亮度插值明暗处理**或**颜色插值明暗处理**技术，它消除了亮度的不连续性。彩图29采用了Gouraud明暗处理技术。虽然大部分在彩图28中的马赫带现象在彩图29中不再可见，如圆环面和圆锥体上的亮脊即是马赫带效应，它是由亮度曲线斜率的陡然变化而不是非连续所引起的马赫带现象。Gouraud明暗处理不能完全消除这种亮度变化。

通过插值多边形顶点的光照值（这些光照值充分考虑到多边形网格所近似的表面），Gouraud 明暗处理技术扩充了施加于单个多边形上的明暗插值概念。Gouraud明暗处理技术过程要求多边形网格的每个顶点的法线是已知的。这可以通过两种方法获取。Gouraud 明暗处理技术可以直接从物体表面的解析描述中计算这些**顶点法线**。另一个方法是，如果网格中没有存储顶点的法向并且又不能直接从真实表面决定，Gouraud建议，可以通过共享该顶点的所有多边形面的法向的平均来近似顶点法线（图14-18）。如果希望一条边可见（如在飞机的机翼与机身的连接处），那么我们可以通过平均该边一侧的多边形的法向找到两个顶点法线，边的两边一边一个。

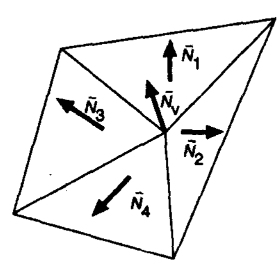
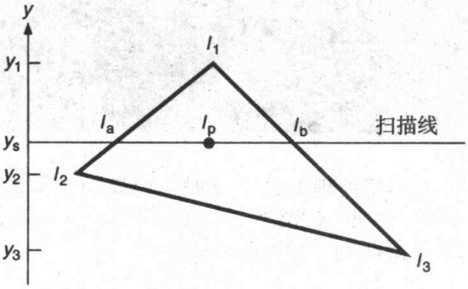


图14-18 可以通过平均规范化的多边形法向量来获取顶点法线。平均法向量 $\bar{N}_v$ 为 $\sum_{1\le i\le n}\bar{N}_i/|\sum_{1\le i\le n}\bar{N}_i|$

下一步，Gouraud明暗处理技术通过将顶点法线运用到所要求的光照模型中找到**顶点的亮度**。最后以类似于13.2节中描述的z值插值方法为整个多边形加上明暗，即先用多边形每条边顶点的亮度值插值出当前扫描线与多边形边交点处的亮度，然后再用交点的光亮值插值出扫描线位于多边形内区段上每个像素处的亮度值（图14-19）。术语Gouraud 明暗处理通常被推广到甚至是独

立的一个多边形的亮度插值的明暗处理，或者是对于与多边形顶点相关的任意颜色值的插值。



$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

图14-19 沿多边形边和扫描线的亮度插值

Gouraud明暗处理沿边的插值可以非常容易地与13.3节中的扫描线可见面算法结合起来。对每条边，我们对每个颜色分量记下起始亮度和亮度在y方向上的单位变化量。可以通过扫描线可见跨度的多边形边的亮度插值对可见跨度进行填充扫描。与在所有的线性插值算法中一样，可以用差分方程来增加效率。

14.2.5 Phong明暗处理技术

Phong明暗处理技术[BUIT75]，又称为**法向量插值明暗处理**技术，是对表面的法向量$\bar{N}$而非亮度进行插值。插值发生在扫描线位于多边形内的跨度上，并且在跨度起始和终点向量之间插值。如果需要，就像在Gouraud明暗处理技术中一样，这些跨度端点处的向量自身又是沿着多边形边通过顶点向量的插值得到的。线性插值可以使用增量法进行计算。向量的三个分量在扫描线之间进行递增。沿扫描线的每个像素，插值所得到的向量首先规格化，并逆映射到世界坐标系或者与之尺度相同的坐标系中，然后用光照模型进行新的亮度的计算。图14-20显示规格化前后的两个边法向量和它们插值得到的法向量。

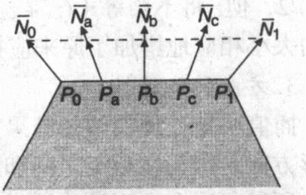


图14-20 法向量插值（来源于[BUIT75]）

彩图30和彩图31分别是用Gouraud明暗处理、Phong明暗处理和一个带有镜面反射项的光照方程生成的。当使用这样的光照模型时，Phong明暗处理大大地改善了Gouraud明暗处理。因为正如图14-21所展示的，Phong模型再现了更真实的高光。想一想，如果在Phong光照模型中的$\cos^n\alpha$光照项的n很大，且其中一个顶点有很小的α值而其周围的顶点却有很大的α时，会发生什么？具有小α值的顶点的亮度相当于高光，但其他顶点却没有高光的亮度。如果使用Gouraud明暗处理，那么，多边形内部的亮度通过该高光亮度和相邻顶点的非高光亮度的线性插值得到，将高光扩展到整个多边形（图14-21a）。将此与图14-21中所示的将线性插值得到的法向量用于计算每个像素的$\cos^n\alpha$项所得到从高光亮度处陡然下落相比较（图14-21b）。而且，如果高光不能落在一个顶点上，那么Gouraud明暗处理可能完全遗漏它（图14-21c），因为没有一个内部顶点可能会比它所插值的顶点更亮。相反，Phong明暗处理允许高光落在多边形内部（图14-21d）。比较彩图30和彩图31中球上的高光。

即使采用不考虑镜面反射的光照模型，因为在每点处都使用法向的一个近似值，所以一般法向插值所得到结果优于亮度插值所得的结果。这减轻了大多数情况下的马赫带效应，但却大大地增加了明暗处理的耗费，因为在光照模型计算时必须对每次插值得到的法向量进行规格化。Duff[DUFF79]提出结合差分方程和表查找来加速这种计算。Bishop和Weimer[BISH86]提出利用泰勒级数展开来对Phong明暗处理进行近似，该方法大大增加了明暗的计算速度。

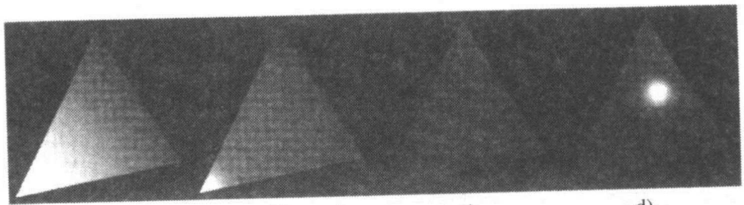


图14-21 使用Gouraud明暗处理和Phong明暗处理的镜面反射光照模型。高光落在左顶点：a)Gouraud明暗处理，b)Phong明暗处理。高光落在多边形内部：c) Gouraud明暗处理，d)Phong明暗处理。（由哥伦比亚大学David Kurlander提供。）

14.2.6 插值明暗处理中的问题

所有的这些插值明暗处理模型都存在许多共同的问题。这里我们罗列了其中的几种。

1. 多边形轮廓

不论一个插值明暗处理模型可以为曲面表面的真实明暗提供多么好的近似，网格的轮廓边表明其仍是多边形的。我们可以通过将表面分成更多的更小多边形来改善这种情况，但相应地也增加了耗费。

2. 透视变形

由于插值计算是透视变化后在三维屏幕坐标系统中进行的，而不是在世界坐标系中完成的，因此产生了变形现象。比如，线性插值使得图14-19中的明暗信息沿着每条边以常值从一条扫描线到另一条扫描线递增。考虑当顶点1比顶点2更远时会发生什么情况。透视缩小意味着扫描线间没有转换的z值的差别将沿着多边形边更远的方向增加。因此，如果$y_s = (y_1 + y_2)/2$，那么$I_s = (I_1 + I_2)/2$，但z_s将不会等于$(z_1 + z_2)/2$。这个问题可以通过采用大量更小的多边形来减轻。减小多边
494~496 形的大小相应地增加了原来应插值而现在进行采样的点的数量，因此增加了明暗处理的准确性。

3. 方向的依赖性

插值明暗处理模型的结果并不独立于多边形的投影方向。因为在顶点之间和水平扫描线上进行插值，所以，当多边形被旋转时，结果可能不同（见图14-22）。当动画相邻帧之间方向缓慢改变时，这个现象相当明显。同样的问题也可能发生在可见面判定中，因为各点的z值是通过各顶点所赋予的z值插值得到的。两个问题都可以通过将多边形分解为三角形来解决（见习题14.2）。此外，Duff[DUFF79]提出了一种插值方法解决了此问题，该方法不需分解且独立于旋转方向，但耗费高昂。

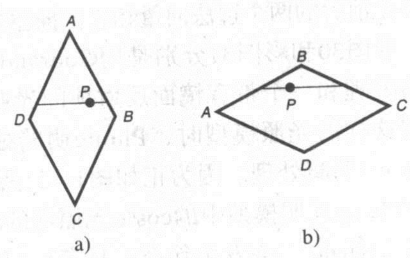


图14-22 在同一多边形不同方向插值推导的点P的不同值。a)中P由点A、B和D插值得到；b)中P由点A、B和C插值得到

4. 在共用顶点处的问题

当两个相邻的多边形并不共用一个落在公共边上的顶点时，明暗处理不连续问题就会发生。考虑图14-23中的三个多边形，其中，顶点C被右边的两个多边形共享，但却不被左边的大多边形所共用。由右边的大多边形直接决定的C点处的明暗信息通常不同于由左边多边形的顶点A、B插值所得C点处的明暗信息。结果，明暗处理就出

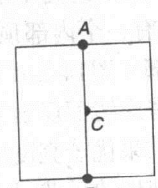


图14-23 顶点C被右边的两个多边形共用，但却不被左边的大多边形所共用

现了不连续。这个不连续可以通过在左边的多边形插入一个额外的顶点来消除，该顶点共享C点的明暗信息。为消除这个问题，可以预先处理静态多边形库，另外一种方法是，如果在事先计算中对多边形（如用BSP树可见面计算）进行分割，那么可以做特别的记号在共用一条被分割的边的边上增加新的顶点。

5. 非直接表示的顶点法线

所计算的顶点法线并不能充分代表了表面的几何性质。比如，如果我们通过共用一个顶点的所有面的法线的平均求取该顶点的法线，那么在图14-24中的所有顶点的法线将相互平行，如果光源很远，就会导致画面的明暗变化很小甚至没有变化。在顶点法线计算前进行此多边形进一步的细分可以解决这个问题。 497

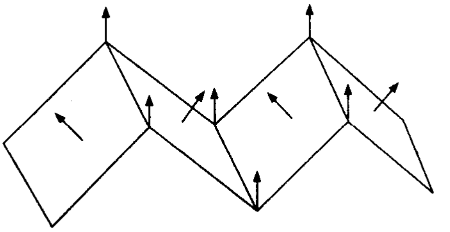


图14-24 计算顶点法线时的问题。所有顶点法线都是平行的

虽然这些问题激发了许多直接处理曲面表面的绘制算法的工作，但多边形却仍是最快且最易进行处理的，这些处理算法构成了大多数绘制系统的核心。

14.3 曲面细节

在平面或双三次曲面上加上我们至今所描述的任何明暗处理模型将产生光滑的均匀曲面——这与我们在现实生活中看到和感觉到的成鲜明对比。下面讨论已有的模拟这些遗漏曲面细节的各种不同算法。

14.3.1 曲面细节多边形

最简单的增加粗糙细节的方法是通过在基多边形上（比如建筑物的一边）覆加*曲面细节多边形*来展示一些特征（如门、窗、平面文字等）。每个曲面细节多边形与它的基多边形共面，并被标上记号以使之在可见面判定时不需与其他多边形进行比较。当为基多边形加上明暗色调时，在它覆盖的区域，曲面细节多边形和它们的材质优先级较高。

14.3.2 纹理映射

当细节越来越精细并且更复杂时，用多边形或其他几何图元进行直接造型不太实际。另外一种方法是将数字化或合成的图像映射至物体表面。这项技术首先由Catmull[CATM74]提出并由Blinn和Newell[BLIN76]进一步改进。这种方法称为**纹理映射**（texture mapping）或**模式映射**（pattern mapping）。图像称为**纹理图**（texture map），它的每个元素通常称为**纹元**（texel）。这个长方形的纹理图落在它自己的(u, v)纹理坐标空间。此外，纹理可以由一个过程定义。彩图34给出了一些用图14-25中的纹理进行纹理映射的例子。在绘制的每个像素中，所选的纹元要么替代该表面的某个材质属性，比如漫射颜色分量，要么对该属性进行去除。一个像素经常被多个纹元所覆盖。为避免走样，必须考虑所有相关的纹元。 498

如图14-26所示，纹理映射可以分为两步完成。简单的方法开始首先将像素的四角点映射到表面上。对一个双三次曲面片，这个映射自然在该表面的(s, t)参数坐标空间中定义了一组点。下一步，在表面的(s, t)参数坐标系下的四个角点被映射到纹理的(u, v)坐标空间。在此纹理图像的四个(u, v)点定义了一个四边形，该四边形在此像素映射的区域近似模拟了原本由表面斜率所描述的更复杂的形状。我们通过落在此四边形内的所有纹元的加权和来计算该像素的值，并以落在此四边形内的纹元所占该像素面积比例加权。如果被转换到(u, v)空间中的点落在纹理图外部，那么就像2.1.3节的图案一样，可以认为纹理图是可以复制的。在(s, t)与(u, v)之间并非总是

使用单一的映射，我们可以定义从0到1的(s, t)空间四边形的四个角与(u, v)空间四边形的对应关系。当一个表面为多边形时，通常将其纹理坐标直接赋予它的顶点。既然在一个任意多边形内部的线性插值所得到的值是依赖于方向的，那么多边形可以首先划分为三角形。然而，即使在三角剖分后，线性插值在透视投影情形下仍然引起变形。这种变形比其他明暗的线性插值所引起的变形更引人注目，因为纹理特征将不能正确地透视缩小。我们可以通过将多边形划分为更小的多边形来获得这个问题的一个近似解，或者以更大的耗费在插值时进行透视分割来获得此问题的精确解。Heckbert[HECK86b]提供了一个所有纹理映射方法的综述。

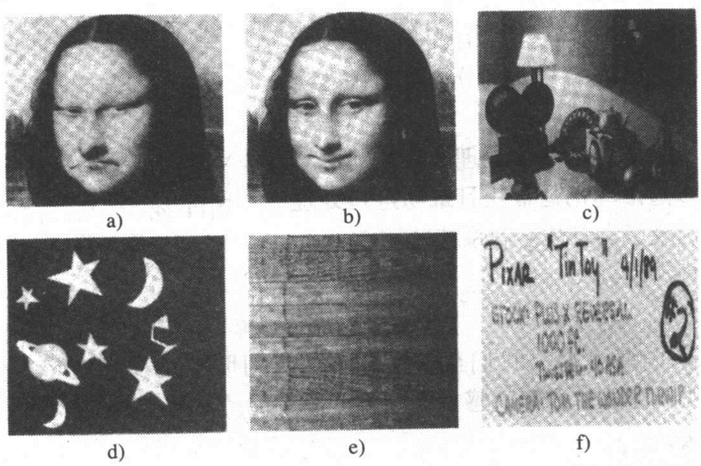


图14-25　用于生成彩图34的纹理。a)皱眉的蒙娜丽莎，b)微笑的蒙娜丽莎，c)油画，d)魔帽，e)地板，f)电影邮票。(Pixar 公司1990版权，由Thomas Williams和H.B.Siegel用Pixar公司的PhotoRealistic RenderMan™软件绘制的图像。)

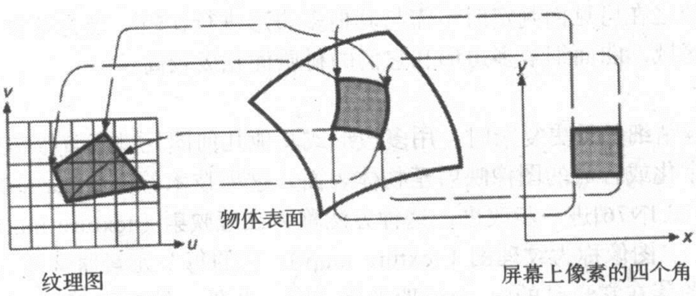


图14-26　从像素到物体表面、纹理图的纹理映射过程

14.3.3　凹凸映射

纹理映射虽然能影响表面的明暗色调，但物体表面仍表现为几何平滑的。如果纹理图是一个粗糙表面的照片，那么加上明暗色调的表面看起来不是很对，因为在生成此纹理图时的光源方向一般不同于照射该表面的光源方向。Blinn [BLIN78b]提出了一种无须显式几何造型就能显示出凹凸不平的表面几何的方法。就像物体表面的细微变化引起物体表面法向扰动一样，该方法在物体表面法向用在光照模型计算前对其进行扰动。这种方法称为**凹凸映射**，是基于纹理映射的一技种术。

一个**凹凸映射**是一组位移量，每个值被用来表示一点比实际表面位置或高或低的位移量。
499~500 凹凸映射的效果非常逼真，观察者往往注意不到物体的纹理并没有影响它的轮廓边。彩图38和彩图39给出了两个凹凸映射的例子。

14.3.4 其他方法

虽然二维映射在大多数情况下是有效的，但它往往产生不了令人信服的结果。当映射至曲面时，纹理经常暴露出其二维本质，并且在纹理拼接处出现许多问题。比如，当一个木纹纹理被映射至曲面物体的表面时，物体看起来似乎是被画上该纹理。为了正确绘制用木头或大理石雕刻出来的物体，Peachey [PEAC85]和Perlin[PERL85]对实体纹理进行了研究。[FOLE90]的第20章描述的这种方法中，纹理是物体上一点位置的一个三维函数。

其他表面性质也可以被映射。比如，Cook已经实现了**位移映射**，在这种方法中，真正的物体表面被偏移，而不仅仅只是物体的表面法线被偏移[COOK84]。该过程必须在可见面判定之前完成。彩图35即用该方法修饰的锥体和圆环的表面。在9.5.1节中我们讨论用分形从简单的几何模型中产生丰富的几何细节。

至此，我们做了一个默认的假设，即对物体上一点的明暗处理过程不受该物体其他部分或其他任何物体的影响。但是实际中一个物体可能被落在它与光源之间的物体所遮蔽；可以透射光，以允许透过它看到另一个物体；或者可以反射其他物体，允许其他物体因为它而被看见。在下面的几节中，我们讨论如何模拟这些效果。

14.4 阴影

可见面算法判定从视点处可见哪些面，而阴影算法判定从光源处可见哪些面。因此，可见面判定算法和阴影算法本质上是一致的。从光源处可见的面不处在该光源的阴影区域中，而从此光源不可见的景物表面则处在该光源阴影区域中。当有多个光源时，必须相对于每个光源对表面进行分类。

这里我们考虑点光源的阴影算法。扩充光源在[FOLE90]的第16章中进行了讨论。从一点光源处的可见性就像从视点处的可见性一样，要么完全可见要么完全不可见。当从光源处看不见面上一点时，该点的光照计算必须加以修正以考虑这种情形。在光照方程中加入阴影效果，则方程变为

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + \sum_{1\le i\le m} S_i f_{att_i} I_{p\lambda_i}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}_i) + k_sO_{s\lambda}(\overline{R}_i\cdot\overline{V})^n] \tag{14-22}$$

其中 501

$$S_i = \begin{cases} 0, & \text{如果在此点光}i\text{被遮挡} \\ 1, & \text{如果在此点光}i\text{没有被遮挡} \end{cases}$$

注意，所有处在点光源阴影处的区域仍为环境光所照射。

虽然计算阴影要求从点光源处计算可见性，但正如我们所指出的，如果不进行任何可见性测试，有可能产生“虚假”的阴影。这可以通过从点光源将所有物体变换为到一预定平面上的多边形投影来高效生成，不必对被变换的多边形所遮挡的表面进行裁剪，也不需检测该变换多边形是否被中间的多边形所遮挡[BLIN88]。这些阴影被用作为曲面细节多边形。但对一般情形，这些“虚假”阴影是不够的，其他许多阴影生成方法也是可能的。我们可以首先进行所有阴影处理，将其与可见性处理进行多种方式的交错处理，或者甚至在可见面处理完成后就进行阴影处理。这里我们考察两类阴影算法。在后面14.7节和14.8节中，讨论了用全局光照方法如何处理阴影。[FOLE90]中概述了另一个阴影算法。为简化解释，如果不做另外的定义，我们认为所有的物体都是多边形的。

14.4.1 扫描线生成阴影算法

最早的阴影生成方法是将扫描线算法加以改进，交错进行阴影和可见面的处理[APPE68；

BOUK70b]。以光源为投影中心，将可能投下阴影的多边形的边投影到与当前扫描线相交的多边形上。当扫描扫过这些阴影边中的一条时，相应地修改该像素点的颜色。

这种算法最简单的实施方法必须为多边形进行 $n(n-1)$ 次到其他多边形的投影计算。Bouknight和Kelley [BOUK70b]代之以一个灵活的预处理步骤。在该步骤中，所有多边形被投影到一个以光源为中心的球面上。两个范围不相重叠的投影可以删除，可以鉴别其他一些特殊情况来限制算法余下部分必须考虑的多边形对的数目。然后，算法将所有这些投影以光源为投影中心向它可能会投下阴影的多边形平面进行投影，如图14-27所示。每个这样的阴影多边形投影都有关于投下阴影或有可能接受这些阴影的多边形的信息。当一个扫描线算法的规则扫描跟踪记录下正扫过的那个规则多边形边时，另外一个单独并行的阴影扫描跟踪记录下当前扫描线正扫过的那个阴影多边形的投影边，因此当前阴影扫描将跟踪记录下所处的那个阴影多边形投影。当计算一个跨度的明暗色调时，如果阴影扫描正处在任何一个投影到该多边形平面的阴影投影中，那么它处在阴影中。因此，在图14-27a中的 bc 段在阴影中，而 ab、cd 段不在。注意，算法不需要以计算阴影的多边形为窗口对阴影多边形的投影进行裁剪。

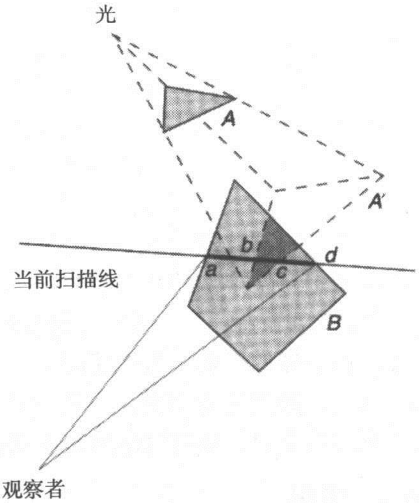


图14-27 使用Bouknight和Kelley方法的扫描线阴影算法。多边形 A 在平面 B 上投下阴影 A'

14.4.2 阴影体

Crow[CROW77]描述了一种如何通过为每个物体产生一个该物体遮挡光源的**阴影体**（shadow volume）来产生阴影的方法。阴影体是由光源和物体所定义的并由看不见的**阴影多边形**所围成。如图14-28所示，相对于光源，物体的每条轮廓边都有一个四边形阴影多边形。该阴影多边形的三个侧面由物体的轮廓边和光源所发出的穿过那条轮廓边的两端点的两条线所决定。每个阴影多边形有一个指向阴影体外的法线。只有那些面对光源的多边形产生阴影体。在Bergeron[BERG86a]所描述的实现中，阴影体（和它的每个阴影多边形）一端由原物体多边形封顶，而另一端由该多边形的一个通过缩放的、法线反向的副本封底。这个通过缩放的副本放在距光的一定距离处，在该距离之后光的衰减能量强度认为是可以忽略不计。我们可以将此距离认作为光的**影响球**。在此影响球之外的任何点实际处于阴影点中而且不需要任何额外的阴影处理。实质上，没有必要为任何完全处在影响球以外的物体产生阴影体。我们通过考虑**影响区域**的方法将此方法进一步推广运用在非均匀发射的光源上，比如，删除落在光源挡板和锥体以外的物体。如果视见体预先已知的话，还可以用视见体对阴影体进一步进行裁剪。该算法将那些封顶多边形也视为阴影多边形。

502~503

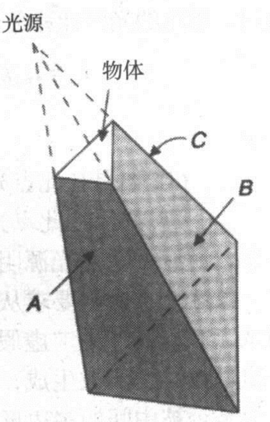


图14-28 阴影体由光源和物体所定义

阴影多边形本身并不被绘制，但却用来决定其他物体是否处于阴影中。相对于观察者，一个正面阴影多边形（图14-28中的多边形 A、B）使得在它之后的物体被投上阴影；一个背面阴影多边形（多边形 C）则忽略了正面的这种效应。考虑从视点 V 到物体上一点的向量。如果与该向量与阴

影体相交的正面多于背面，该点处于阴影中。因此，在图14-29a中的点*A*和*C*处于阴影中。这是惟一的一种当*V*不在阴影中时点在阴影中的情形。因此点*B*被照亮。如果*V*处在阴影中，则存在另外一种点在阴影中的情形，即从视点到物体上一点的向量没有遇到遮挡该视点的物体多边形的相应背面阴影多边形。因此，图14-29b中的点*A*、*B*、*C*皆处在阴影中，即使从*V*到*B*的向量如它在图14-29a中所经历的一样与相同数量的正面阴影边界多边形与背面阴影边界多边形相交。

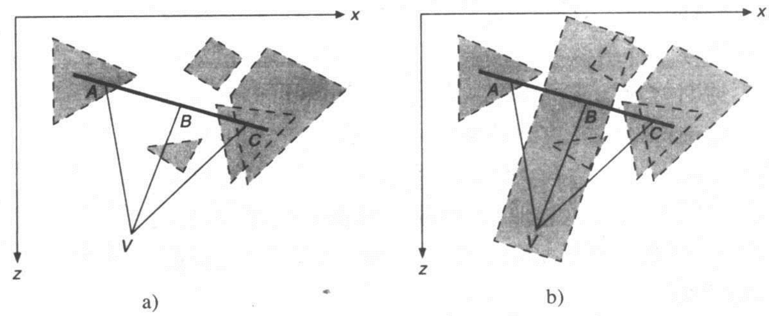


图14-29 为位于*V*的观察者决定一点是否在阴影中。虚线表示阴影体（加上阴影）。a)*V*不在阴影中，点*A*和*C*在阴影中，但点*B*被照亮；b)*V*在阴影中，点*A*、*B*和*C*都在阴影中

我们可以通过为每个正面（相对于观察者）阴影多边形赋值为+1，而背面阴影多边形为-1来计算一个点是否处于阴影。设置一个计数器，该计数器的初始值设置为包含当前视点的阴影体的数目，并且按视点与物体上的点之间的所有阴影多边形的相关值递增。如果该点的计数器的值为正，则该点处于阴影中。对每个视点，只需要计算一次包含该视点的阴影体的数目，通常取由视点到无穷远处的一个任意投射线所交的所有阴影多边形的值的和的负数。 504

多个光源可以通过为每个光源单独建立阴影体来处理，为阴影体的每个阴影边界多边形标上光源标识并为每个光源保持一个不同的计数器。Brotman和Badler[BROT84]实现了*z*缓存的阴影体算法，Bergeron[BERG86a]讨论了有效地处理了包括非平面多边形的任意多面体物体的扫描线阴影体算法的实现。Chin和Feiner[CHIN89]描述了一种对象精确的算法。该算法用10.6.4节中讨论的BSP树实体造型表示为多边形环境建立单一的阴影体。

14.5 透明性

正如物体表面会产生镜面反射或漫反射那样，透明或者半透明的物体则可以透光，我们可以透过诸如玻璃这类**透明**材质观看，尽管大多数光线已被折射（弯曲），而对于像磨砂玻璃的**半透明**材质则产生了漫射透射。由于半透明物体表面以及内部的不规则性，穿过该半透明材质的光线将会变得杂乱无章，这样透过半透明材质所看到的物体必然显得模模糊糊。

14.5.1 无折射的透明性

模拟透明性的最简单方法是忽略折射，这样一来，光线在穿越物体表面时不再被弯曲。因此，穿越透明物体视线上所见的物体也必然在几何意义上落在该视线上。虽然无折射的透明性并不存在，但它通常产生比折射更为有用的效果。例如，它可以体现一种透过物体表面后无扭曲的视图。正如我们以前曾经提到的那样，完全的照片真实感并不总是我们生成画面的目标。

通常有两种方法来近似模拟透过一个物体看见另一个物体时发生的物体颜色的混合的方法。我们分别称之为**插值透明法**和**滤光透明法**。

1. 插值透明法

如图14-30所示，考虑在不透明的多边形2以及观察者中间存在一个透明的多边形1时会发

生什么。**插值透明法**通过对两个多边形各自的计算所得的明暗值进行线性插值来决定位于两个多边形重叠部分的像素的明暗值：

$$I_\lambda = (1 - k_{t_1})I_{\lambda_1} + k_{t_1}I_{\lambda_2} \tag{14-23}$$

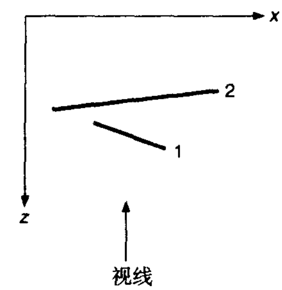


图14-30 两个多边形的截面图

透射系数k_{t1}用来度量多边形1的**透明度**，取值为0到1。当k_{t1}为0时，多边形1变成了不透明的，不再透过光线；而当k_{t1}取值为1时，多边形1便成了完全透明的，对I_λ值不再有任何影响：（$1 - k_{t1}$）值称为多边形的**不透明度**。插值法可以理解为将一个多边形模拟为一
505 个非常精细的不透明材料构成的网格。透过这个网格可以看到其他物体：k_{t1}值是透过该网格能看到其他物体的比例。这样处理的一个完全透明的多边形将不会有任何镜面反射。为了达到更真实的效果，我们可以首先只对多边形1的环境光和漫射光部分以及多边形2的全部的明暗色调进行插值，然后再加入多边形1的镜面反射光部分[KAY79b]。

另一种方法通常称为**筛子透明法**。它通过只绘制那些与透明物体的投影相联系的像素来生成网格。像素的（x, y）地址的低位被用于索引，指向一个透明位掩码。如果这个索引位值为1，则写出该像素值。否则，该像素被挂起，下一个最接近的物体成为可见。

2. 滤光透明法

滤光透明法将多边形看成是一个有选择地允许一定波长光线通过的透明滤镜，模型可表示为

$$I_\lambda = I_{\lambda_1} + k_{t_1}O_{t\lambda}I_{\lambda_2} \tag{14-24}$$

这里$O_{t\lambda}$是多边形1的**透明度颜色**。可以通过给不同的波长λ选择不同的$O_{t\lambda}$值模拟一个颜色滤镜。无论在插值透明法还是在滤光透明法，若在这些多边形前还有别的透明多边形，那么运算将从后向前递归地执行，并把上一次计算所得的I_λ作为当前的$I_{\lambda 2}$。

3. 实现透明效果

可以非常容易地将一些可见面算法加以修改加入透明效果。如扫描线算法和列表优先级算法。在列表优先级算法中，读入一个将被透明多边形覆盖的像素的颜色值，并且在扫描转换该多边形时，该值将用于光照明模型的计算中。

大部分基于z缓存的系统支持筛子透明法，因为该算法允许透明物体和不透明物体混合在一起并以各种顺序进行绘制。但是要想把式(14-23)和式(14-24)所示的透明度引入到z缓存算法却比较困难，因为多边形是以随意的顺序绘制的。想像一下在先处理了几个叠加的透明多边形后，接着是一个不透明的的多边形。我们当然希望将此多边形插入相应的透明多边形后。但是，在z缓存中并没有存储所需的信息来判定这些多边形的颜色值。一个简单的但却不正确的方法是：最后绘制透明多边形，并将它们的颜色与帧缓存中已有的颜色值结合，但并不修改z缓存，然而，当两个透明多边形重叠时，它们的相对深度并没有得到考虑。

Kay 和Greenberg[KAY79b] 实现了一种模拟发生在薄曲面轮廓边界上的光的不断衰减现象的方法。在这些地方，光线通过更多的材质。它们将k_t定义为一个在透视投影变换后曲面法向的z分量值的非线性函数。

506 $$k_t = k_{t_{min}} + (k_{t_{max}} - k_{t_{min}})(1 - (1 - z_N)^m)$$

其中k_{tmin}和k_{tmax}是物体透明度的最小值和最大值，z_N是计算k_t的那个点的规格化的曲面法向量的z分量值，m是一个幂因子（通常为2或3）。m值越高，模拟的曲面越薄。这里算出的k_t可以作为式(14-23)或式(14-24)中的k_{t1}。

14.5.2 折射透明性

折射透明性的模拟要比无折射的透明性的模拟困难得多，因为几何视线与光学视线不同。在图14-31所示的情况下，如果考虑折射，沿所示视线透过透明物体后可看见A；若忽略折射，则应该看见B。Snell法则给出了入射角θ_i和折射角θ_t的关系：

$$\frac{\sin\theta_i}{\sin\theta_t}=\frac{\eta_{t\lambda}}{\eta_{i\lambda}} \tag{14-25}$$

其中，$n_{i\lambda}$和$n_{t\lambda}$分别为光透过材料的**折射率**。一种材料的折射率是光线在真空中传播的速度与光在该材料中传播速度的比值，随光的波长甚至温度而变化。真空的折射率为1，空气的折射率则接近于1；所有其他材料有着更高的值。折射率依赖于波长。在许多折射实验中我们都可以明显地观察到这个事实，如我们所熟悉的但却很难模拟的**色散**（dispersion）现象，一种光被折射为光谱的现象[THOM86; MUSG89]。

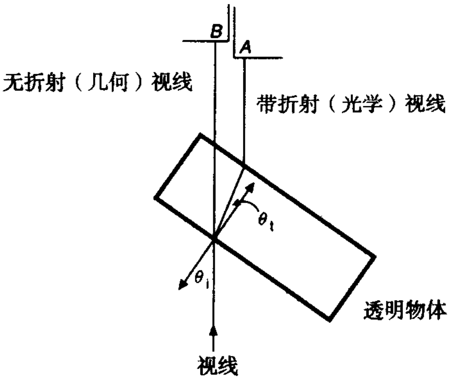


图14-31 折射

1. 折射向量的计算

可以由下式求得折射方向的单位向量$\overline{T}$：

$$\overline{T}=\sin\theta_t\overline{M}-\cos\theta_t\overline{N} \tag{14-26}$$ 507

其中$\overline{M}$是在向量$\overline{I}$和$\overline{N}$所在平面上的垂直于$\overline{N}$的单位向量[HECK84]（图14-32）。回忆一下在14.1.4节中我们利用$\overline{S}$来计算折射向量$\overline{R}$，我们有$\overline{M}=(\overline{N}\cos\theta_i-\overline{I})/\sin\theta_i$。通过替换，

$$\overline{T}=\frac{\sin\theta_t}{\sin\theta_i}(\overline{N}\cos\theta_i-\overline{I})-\cos\theta_t\overline{N} \tag{14-27}$$

若令$\eta_{r\lambda}=\eta_{i\lambda}/\eta_{t\lambda}=\sin\theta_t/\sin\theta_i$。重排各项后有

$$\overline{T}=(\eta_{r\lambda}\cos\theta_i-\cos\theta_t)\overline{N}-\eta_{r\lambda}\overline{I} \tag{14-28}$$

注意，$\cos\theta_i$是$\overline{N}\cdot\overline{I}$，$\cos\theta_t$可以由下式求得：

$$\cos\theta_t=\sqrt{1-\sin^2\theta_t}-\sqrt{1-\eta_{r\lambda}^2\sin^2\theta_i}=\sqrt{1-\eta_{r\lambda}^2(1-(\overline{N}\cdot\overline{I})^2)} \tag{14-29}$$

因此有

$$\overline{T}=\left(\eta_{r\lambda}(\overline{N}\cdot\overline{I})-\sqrt{1-\eta_{r\lambda}^2(1-(\overline{N}\cdot\overline{I})^2)}\right)\overline{N}-\eta_{r\lambda}\overline{I} \tag{14-30}$$

2. 全反射

当光线从一个高折射率介质进入一个低折射率介质时，光线透射角θ_t将大于入射角θ_i。如果θ_i足够大，以至于θ_t超过了90°，则光线则会在介质之间的交界面上反射回来，而不是进入另一个介质。这种现象称为**全反射**。发生全反射的θ_i的最小值称为**临界角**。可以很容易观察这种现象，如透过鱼缸观察您的手。当您的观察角大于临界角时，您所能看到的就仅仅只是那些紧贴在玻璃上的部分手掌，因为在这部分手掌和玻璃之间没有空气层（空气的折射率要小于玻璃和水）。当$\sin\theta_t$为1时，θ_i的值便是临界角度值。若在式(14-25)中将$\sin\theta_t$置为1，临界角即为$\sin^{-1}(\eta_{t\lambda}/\eta_{i\lambda})$。当式(14-30)中的那个平方根得到一个虚数时，全反射现象便发生了。

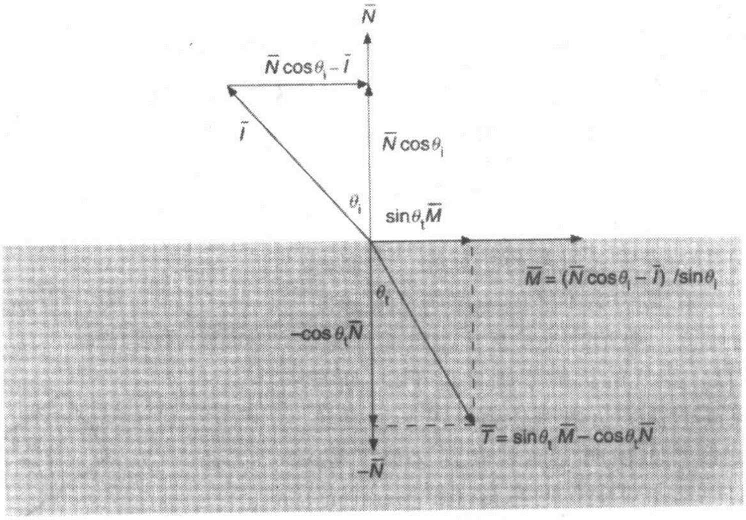


图14-32 计算折射向量

14.7节讨论了在用光线跟踪来模拟折射透明效果时Snell法则的用法。

14.6 全局光照算法

光照模型以光源直射光和经由一点所在的表面及其他表面的反射和透射的光共同决定该点的颜色，这种非直接反射和透射的光通常被称为**全局光照**。与此对应，**局部光照**是那些直接从光源到达为其计算明暗的点的光。我们用环境光项近似全局光照，环境光被认为对于所有物体上的所有点是不变的，不依赖该物体或观察者位置，也不受近旁可能阻隔环境照明光线的物体的影响。此外，我们已经通过阴影和透明性观察到了一些有限的全局光照效果。

在现实环境中，多数光线并不是直接来自光源。有两种不同的强调全局光照作用的算法被用来生成图像。14.7节讨论可见面光线跟踪算法的扩充，以交错进行可见面的判定和明暗计算来描述阴影、反射和折射。因此全局镜面反射和透射为物体表面补充了局部镜面光照、漫射光照和环境光照。与此对应，在14.8节讨论的各种辐射度算法完全将可见面判定和明暗计算分开，首先以一个独立于视点的阶段模拟环境与光源的所有相互作用，然后才使用传统可见面判定和插值明暗处理为各目标视点生成不同的画面。

依赖于视图的算法（如光线跟踪算法）和独立于视图的算法（如辐射度算法）的差别是很重要。**依赖于视图**的算法在给定视线方向离散视图平面成点集并在这些点计算光照方程以决定这些点的光照。与之相对应的是，**独立于视图**的算法将环境表面离散，为任何点任意视线方向的光照方程计算提供足够的信息。依赖于视点的算法适合处理与观察者位置密切相关的镜面反
508~509 射现象，但在模拟图像大片区域变化不大或不同视点方向图像变化不大的漫射现象需要进行大量额外计算。另一方面，独立于视点算法对于处理漫射现象非常有效，但需要大量存储资源以获取与镜面反射有关的足够信息。

最终，所有这些方法都试图解决Kajiya[KAJI86]所指的**绘制方程**，该绘制方程根据光从一点照射到另一点的亮度以及从其他所有照射到第一点并被反射到第二点的光的亮度，来表示光从一点到另一点的转换。同样，从其他所有点转换到第一点的光可以递归由该方程表示，Kajiya将此绘制方程表示为

$$I(x, x') = g(x, x')\left[\varepsilon(x, x') + \int_S \rho(x, x', x'')\, I(x', x'')dx''\right] \quad (14\text{-}31)$$

其中，x, x', x''是环境中的点。$I(x, x')$是从x'到x的亮度。$g(x, x')$是几何形状项，当x与x'相互不可见时为0，可见时为$1/r^2$，这里r是它们之间的距离。$\varepsilon(x, x')$是指由x'发射到x的光的亮度。在视点处x的$g(x, x')\ \varepsilon(x, x')$的初始化计算即完成了$x$点球面上的可见面判定。方程对所有面$S$上的所有点进行积分。$\rho(x, x', x'')$是从$x''$反射并经由$x'$到达$x$的光的亮度（包括镜面反射和漫反射）。因此，绘制方程表明从x'到达x的光包括了它自身发射的和它所反射的从其他表面来的光，这些面自身发光并且递归地反射从其他表面来的光。

如我们所见，判别解绘制方程的方法是否有效，一定程度上取决于该方法如何处理这些剩余项和递归方法，它所支持的漫反射和镜面反射是怎样结合上，以及取决于它如何模拟曲面之间的可见关系。

14.7 递归光线跟踪

这一节中，我们将把基本光线跟踪算法（在13.4节讨论的）推广到处理阴影、反射和折射中去。该基本光线跟踪算法使用先前介绍的任一光照模型为与视线相交的最近一点的物体计算其所在像素的颜色。在计算阴影时，我们发出一条从该交点到每个光源的一条新的射线。图14-33中显示了只有一个光源的情况下的情形，该图引自Appel的文章[APPE68]，该文章是计算机图形学领域发表的有关光线跟踪算法的第一篇论文。如果这一束**阴影光线**中的任一条在该光线上与一物体相交，那么在该点处该物体处于阴影中，并且明暗处理算法将忽略该阴影射线所对应的光源。

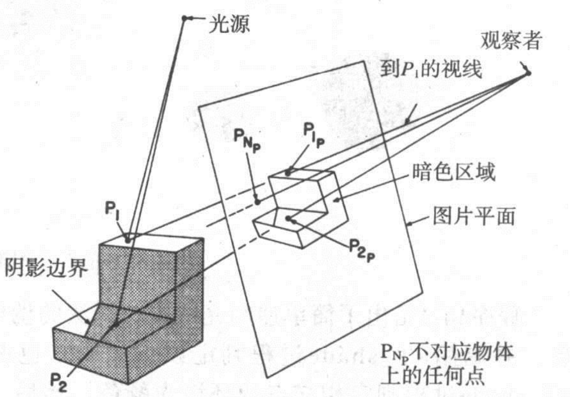


图14-33 确定物体上一点是否在阴影中。（由IBM T.J. Watson 研究中心Arthur Appel提供。）

由Whitted [WHIT80]和Kay [KAY79a]开发的光照模型从根本上把光线跟踪推广到镜面反射、折射透明性。彩图41就是早期应用该算法生成的。除阴影线，Whitted递归光线跟踪算法从相交点处有条件地生成出**反射线**和**折射线**，如图14-34所示。这些阴影线、反射线和折射线通常称成**次级光线**，以区别从眼中发出的**主光线**。如果物体是产生镜面反射的物体，那么反射线相对于该表面的法向以方向为$\bar{R}$反射出来，该反射方向的计算可参见14.1.4节。如果物体是透明的且不发生任何内部反射，那么折射线将沿着T方向以Snell定律所决定的折射角进入该物体，Snell定律详见14.5.2节（注意您的入射光线可能与这些小节中的入射光线方向相反）。

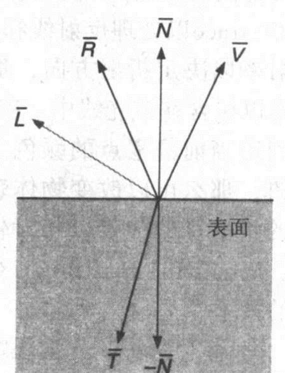


图14-34 从相交点所产生的反射线、折射线和阴影线

如图14-35所示，每条反射线和折射线可能依次递归地产生阴影线、反射线和折射线。这些线就形成图14-36所示的**光线树**。在Whitted的算法中，在镜面反射线和折射线不再与任何物体相交时，或达到用户定义的光线跟踪最大深度或系统已

耗尽所有存储资源时，光线停止跟踪，光线树由底而上地进行估算，树中的父节点的亮度由子节点的亮度计算得来。

Whitted光照方程可表示如下

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{att_i} I_{p\lambda_i} [k_d O_{d\lambda} (\overline{N} \cdot \overline{L}_i) + k_s (\overline{N} \cdot \overline{H}_i)^n] + k_s I_{r\lambda} + k_t I_{t\lambda} \quad (14\text{-}32)$$

510~511 其中，$I_{r\lambda}$是反射线的亮度，k_t是在0~1范围内取值的**透射系数**。$I_{t\lambda}$是折射光线的亮度。$I_{r\lambda}$和$I_{t\lambda}$的值通过递归地用式(14-32)对反射光线和折射光线相交的最近点进行计算来决定。Whitted用所计算得到的光线的I_r乘以光线所经路程长度的倒数来近似计算亮度随距离的衰减。Whitted将S_i表示为该阴影线所交物体的k_t的连续函数，而不是将其作为一个delta函数（如在式(14-22)中）。因此一个透明的物体在它投下阴影的那些点处将比非透明物体遮挡少一些光。

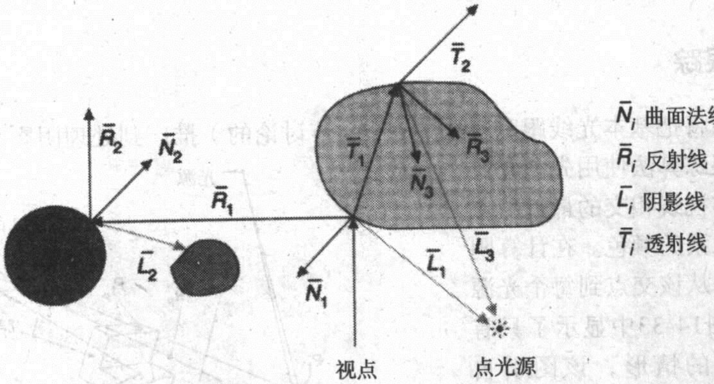


图14-35 光线递归地产生其他光线

程序14-1给出了简单递归光线跟踪算法的伪代码。RT_trace过程判定与一光线相交的最近点，并调用RT_shade过程判定该点的明暗色调。首先RT_shade过程判定相交点的环境光颜色。然后，向所有位于在为其计算明暗的表面的一边的光源发出阴影光线以决定该光源对颜色的贡献。非透明物体完全遮挡了光，而透明物体将调节光的亮度，如果我们在光线树上所跟踪的深度不太深，那么为反射物体和透明物体递归地调用RT_trace以处理反射线和折射线。因为需要两种介质的折射率以决定折射方向，那么光线所穿过的物质的折射率可以包含在该光线中。RT_trace过程只将光线树保存到能判定当前像素点的颜色。如果可以为整个图像保存光线树，那么可以改变物体表面的特性，并且可以以光线树的重估算的代价以相对较快的速度重新计算一幅新图像。Sequin和Smyrl[SEQU89]给出了一些大大缩短了光线树的计算时间和减少光线树的存储空间的技术。

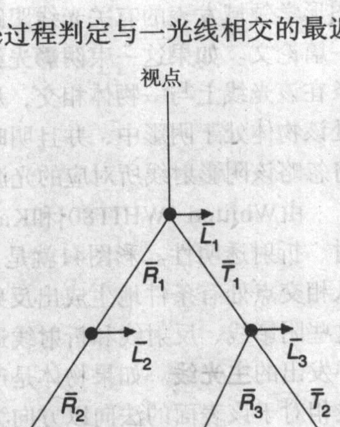


图14-36 图14-35的光线树

图14-35给出了光线跟踪算法处理折射时的一个基本问题，阴影光线$\overline{L}_3$并不以它到光源的路线折射。事实上如果我们以其穿过该大物体的点的方向将$\overline{L}_3$进行折射，则它不会到达光源。此外，在决定一条光线的折射路径时，只为每条光线使用了一个折射率。

程序14-1 没有反走样效果的简单递归光线跟踪算法的伪代码

```
选择投影中心以及视图平面上的窗口;
for ( 图像的每条扫描线 ) {
    for ( 扫描线上的每个像素pixel ) {
        确定从投影中心穿过该像素pixel的光线ray;
        pixel = RT_trace (ray, 1);
    }
}

/* 计算与物体的交点并计算最近交点的光照值 */
/* depth是光线树的当前深度 */

RT_color RT_trace ( RT_ray ray, int depth )
{
    确定光线与某个物体object的最近交点intersection;
    if ( 命中物体 ) {
            计算交点处物体的法向normal;
            return RT_shade (closest object hit, ray, intersection, normal, depth);
    }
    else
        return BACKGROUND_VALUE;
}

/* 计算物体上一点的光照值，跟踪阴影光线、反射光线和折射光线 */

RT_color RT_shade (
        RT_object object,               /* 相交的物体 */
        RT_ray ray,                     /* 入射光线 */
        RT_point point,                 /* 与其光照值的交点 */
        RT_normal normal,               /* 交点处的法向 */
        int depth )                     /* 所处光线树的深度 */
{
RT_color color;                 /* 光线的颜色 */
RT_ray rRay, tRay, sRay;        /* 反射光线，折射光线，阴影光线 */
RT_color rColor, tColor;        /* 反射光线颜色和折射光线颜色 */

    color = 环境光项;
    for ( 每个光源 ) {
        sRay = 从交点到光源的光线;
        if ( 法向与到光源的方向的点积为正 ) {
            计算被不透明物体和半透明物体遮挡的量,
            并用该值乘漫射项和镜面反射项，然后将其加入 color;
        }
    }
    if ( depth<最大深度 ) {         /* 如果深度太深返回 */
        if ( object是反射物体 ) {
            rRay = 从交点point向反射方向发射的光线;
            rColor = RT_trace (rRay, depth + 1);
            将rColor与镜面反射系数相乘并加入color;
        }
        if ( object为透明物体 ) {
            tRay = 从交点向透射方向发射的光线;
            if ( 没有发生全反射 ) {
                tColor = RT_trace (tRay, depth + 1);
                将tColor与透射系数相乘，并加入color;
            }
        }
    }
    return color;       /* 返回光线的颜色值 */
}
```

光线跟踪算法特别容易产生由于有限数值精度所引起的问题。当我们计算与次级光线相交的物体时，这些问题就突现出来了。在计算可见物体上与光线相交点的x, y, z的坐标后，x, y, z坐标被用来定义次级光线的起始点，并为其确定参数t（见13.4.1节）。如果刚刚与光线相交的物体又和一新光线相交，由于数值精度的限定，该物体往往会有一个小的、非零的参数t。如果不加处理，那么这种错误的相交点将导致视觉误差。例如，如果该光线是阴影线，那么物体可被认为它自己挡了光，导致不正确的“自阴影”的斑点表面。对这种由阴影线导致的问题的一种简单解决方法是把生成次级光线的物体当成一个特殊情况，不在其表面上进行相交测试。当然，当物体确实支持自身遮挡或者透射光线穿过一个物体并为在该物体内部被反射时，这种处理不再可行。有一个更普通的解决方法，该方法为每次相交计算abs(t)并将其与一个小的误差容忍值相比较，当abs(t)低于该误差容忍值时，就舍弃它。

Whitted发表在*SIGGRAPH'79*[WHIT80]上的论文和他用所描述的算法所做的电影重新引起了人们对光线跟踪算法的兴趣。递归光线跟踪算法带来令人印象深刻的效果，例如阴影、镜面反射和折射透明效果等这些先前很困难也不可能获得的效果。此外，一个简单光线跟踪算法十分容易实现。因此人们投入了大量的努力致力于提高该算法的效率和图像的质量。更多的细节请参见[FOLE90]的16.12节及 [GLAS89]。

14.8 辐射度方法

尽管光线跟踪方法成功地模拟了景物表面的镜面反射和规则折射的透明效果，但该方法利用无方向的环境光代替所有其他光照的贡献。辐射度方法对热辐射的发散和反射的描绘基于热能工程模型，它为物体间的多重反射提供了一种更加精确的处理方法，避免了对漫射光线直接
512~514 处理。算法首先由Goral、Torrance、Greenberg和Battaile [GORA84]以及Nishita和Nakamae [NISH85]提出，这些算法假设光能在一个封闭的环境中是守恒的。每一个表面发出或反射的能量被说明为从其他表面反射或被其他表面吸收的能量。单位时间里离开一个表面的能量称为**辐射度**，是该表面释放的能量以及其反射或透射从其本身或其他表面来的能量的总和。因此，计算一个环境中表面辐射度相关的方法被称为**辐射度方法**。与通常的绘制算法不同，辐射度方法首先以与视点无关的方式计算光在环境中的相互作用，然后只需要可见面判定和明暗插值的开销就可绘制一个或多个视图。

14.8.1 辐射度方程

在以前的明暗处理算法中，光源总是与它所照射的表面分开处理。与之相反，辐射度方法允许任何表面发射光线，因而，所有光源一致地被表示为具有面积。设想把场景划分为有限数量的离散面片n，每一面片大小固定并且在该表面上均匀发射和反射光。如果把每一个面片看成是一个不透明的朗伯漫射的发射体和反射体，则对于表面i，

$$B_i = E_i + \rho_i \sum_{1\leqslant j\leqslant n} B_j F_{j-i} \frac{A_j}{A_i} \tag{14-33}$$

B_i、B_j分别为面片i和j的辐射度，以能量／单位时间／单位面积（即W/m^2）度量。E_i是其自身发射的能量，与辐射度具有相同的度量单位；ρ_i是面片i的反射系数，是一个无度量单位的量；F_{j-i}是无度量单位的**形状因子或构造因子**，用来表示从面片j辐射并到达整个面片i的那部分光能，该因子的设定须将两个面片的形状以及相对方向和遮挡物的存在考虑在内。A_i和A_j是面片i和j的面积。

式(14-33)表明了离开物体表面单位面积的能量是辐射光和反射光之和。反射光由所有入射光之和乘以反射率计算得到，而入射光则是场景中离开所有面片的光的和乘以到达接受面片单

位面积的比率。$B_j\ F_{j-i}$是离开面A_j单位面积以及到达A_i的所有光能，因而，乘以一个面积比A_j/A_i就决定了所有离开A_j以及到达A_i单位面积的光能。

自然地，漫射环境中的形状因子之间存在着一个简单的相互关系：

$$A_iF_{i-j}=A_jF_{j-i} \tag{14-34}$$

所以，式(14-33)可以简化为 515

$$B_i = E_i + \rho_i \sum_{1\leqslant j\leqslant n} B_j F_{i-j} \tag{14-35}$$

移项有

$$B_i - \rho_i \sum_{1\leqslant j\leqslant n} B_j F_{i-j} = E_i \tag{14-36}$$

因此，环境中面片间光能的相互作用可被表示为一组联立方程：

$$\begin{bmatrix} 1-\rho_1F_{1-1} & -\rho_1F_{1-2} & \cdots & -\rho_1F_{1-n} \\ -\rho_2F_{2-1} & 1-\rho_2F_{2-2} & \cdots & -\rho_2F_{2-n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ -\rho_nF_{n-1} & -\rho_nF_{n-2} & \cdots & 1-\rho_nF_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_n \end{bmatrix} \tag{14-37}$$

注意，必须计入一个面片对它自身反射光能的贡献（例如，面片可能是凹的）。所以，通常情况下，联立方程组对角线上的每一个因子并不是1。联立方程组(14-37)应该对所有光模型中考虑的每个波段进行求解，因为ρ_i和E_i都是依赖于波长的。但是形状因子是独立于波长的，并且是严格的几何函数，因而在光线及表面折射率改变的时候不需重新计算。

联立方程组(14-37)可用Gauss-Seidel迭代法求解[PRES88]，生成每一个面片的辐射度值。然后这些面片可用传统的可见面算法从任何当前视点进行绘制。计算得到的各面片各波段的辐射度就是该面片的亮度。不用微面元明暗处理方法，我们可以通过顶点所在面片辐射度值计算得到顶点的辐射度，以对面片进行插值明暗处理。

Cohen和Greenberg [COHE85]提出用以下方法来确定顶点辐射度：如果顶点为一个表面内部顶点，则共享该顶点的所有面片的辐射度的均值即为该顶点的辐射度值；如果该顶点在边界上，则找到最近的内部顶点v，该边界顶点的辐射度与B_v的均值应该是共享这个边界顶点的那些面片的辐射度的均值。考虑图14-37中的面片。内部顶点e的辐射度为$B_e=(B_1+B_2+B_3+B_4)/4$。边界顶点b的辐射度是通过找到最近的内部顶点e来计算。注意b被面片1和2所共享。因而为确定B_b，利用上述定义，$(B_b+B_e)/2=(B_1+B_2)/2$求取，得到$B_b=B_1+B_2-B_e$。a的最近内部顶点也是e，而a仅仅只是面片1的一部分，所以（B_a+B_e）$/2=B_1$，因此$B_a=2B_1-B_e$。其他顶点的辐射度计算与此类似。

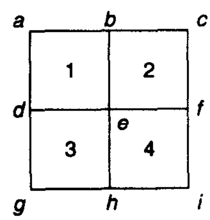


图14-37 用面片辐射度计算顶点辐射度

最初的辐射度方法由Goral等人实现[GORA84]。他用轮廓线的积分为无封闭曲面的凸环境计算精确的形状因子，如彩图43所示。注意，由于相邻表面间的漫反射正确的“渗色”效果在 516
模型和绘制图像中都是可见的。为使辐射度方法能够实用，首先应开发计算封闭的曲面之间的形状因子的方法。

14.8.2 计算形状因子

Cohen和Greenberg[COHE85]采用了一种图像精度的可见面算法来近似遮挡曲面间的形状因子。考虑图14-38所示的两个面片，从微分面元dA_i到微分面元dA_i的形状因子为：

$$dF_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j \tag{14-38}$$

对于图14-38中微分面元dA_i和dA_j之间的光线，θ_i是光线与A_i法向量的夹角，θ_j是该光线与A_j法向量的夹角，r是该光线的长度。H_{ij}取0或1，取决于从dA_i是否可看见dA_j。为求从微分面元dA_i到有限面积A_j的形状因子F_{di-j}，需要在面片j上进行积分，因此，

$$F_{di-j} = \int_{A_i} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j \tag{14-39}$$

最后，A_i到A_j的形状因子是式(14-39)在面片i上的面积均值：

$$F_{i-j} = \frac{1}{A_i} \int_{Ai} \int_{Aj} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j\, dA_i \tag{14-40}$$

如果假设一个面片上的中心点代表了该面片上其他的点，那么F_{i-j}可以用计算面片i中心的微分面元dA_i的形状因子F_{di-j}来近似。

Nusselt [SIEG81]讨论了可以用投影来计算F_{di-j}，即把A_j的对于dA_i可见的部分投影到以dA_i为中心的单位半球面上，然后把投影区域再垂直投影到半球的底面上，再除以底面面积（图14-39）。其中投影到单位半球面等价于计算式(14-39)中的$\cos\theta_j/r^2$，投影到底面则相应于乘以$\cos\theta_i$，除以单位圆的面积则是考虑到分母中的π。

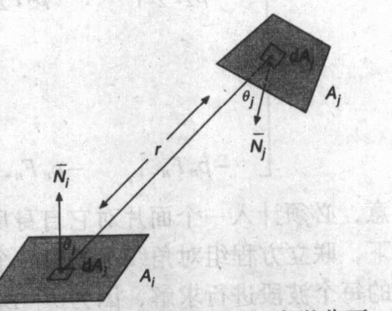


图14-38 计算一个面片和一个微分面元间的形状因子

除了分析上将每一个A_j投影到半球面上，Cohen和Greenberg开发了另一种有效的图像精度算法，该算法将A_j投影到以dA_i为中心的立方体的上半部分。立方体的顶面是与表面A_i平行的（如图14-40所示）。这个**半立方体**的每个面都被划分为许多大小相等的方形单元（本书图中所用的分辨率从50×

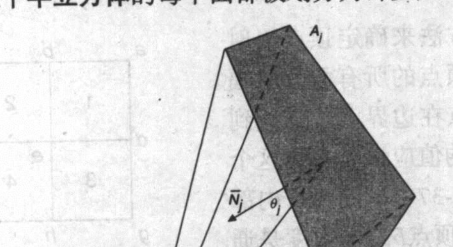


图14-39 用Nusselt的方法确定一个面片和一个微分面元间的形状因子。半球底面上的投影面积和底面面积的比值就是形状因子

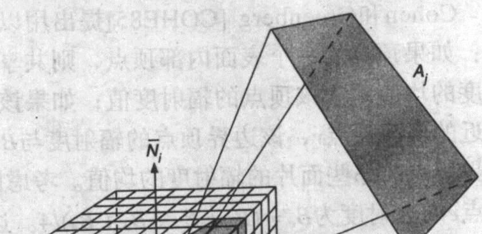


图14-40 半立方体是以面片为中心的正方体的上半部分。（选自[COHE85]。）

50到数百不等）。其他所有面片被以该立方体的中心和它的五个面所定义的视见体所裁剪，并且每个被裁剪的面片都被投影到该半立方体的相应面上。可用**分项缓存**算法[WEGH84]来记录每个单元上最近的相交面片标识。对半立方体的每面执行z缓存算法，记录每一单元最靠近的面片id而不是深浅，就可计算出这些分项缓存。半立方体的每个单元与该位置的Δ形状因子有关。对于任一面片j，其F_{di-j}可以用包含面片j的id的所有半立方体单元Δ形状因子值的和来近似。

因为半立方体算法中大多数的计算都要用到计算分项缓存，所以可以利用现有的z缓存硬件。另一方面，因为所有操作都是图像精度的，所以半立方体很容易产生走样。

14.8.3 逐步求精算法

考虑到至此所描述的辐射度求解过程的代价都很高，那么是否可以逐步地逼近该算法的结果呢？是否可以先产生一些虽然不精确但是有用的图像，可以利用它随计算时间的增加逐步提高精度呢？上几节中描述的辐射度方法不能这样做，原因有二：一是必须进行完整的Gauss-Seidel迭代后才能再对面片辐射度进行计算；二是在最初计算所有面片之间的形状因子并将其保留到计算结束，需要$O(n^2)$的时间和空间。Cohen、Chen、Wallace和Greenberg[COHE88]开发了一种逐步求精的辐射度算法解决了上述两个问题。

考虑至今所描述的辐射度方法。式(14-37)第i行的计算为面片i的辐射度B_i提供了一种估计，B_i由式(14-35)表示，该方程基于对其他面片辐射度的估计的。式(14-35)中求和的每一项代表面片j对面片i的辐射度的影响： 517~519

$$B_i\text{归因于}B_j = \rho_i B_j F_{i-j}\text{，对所有的}j \tag{14-41}$$

因而，这一方法“收集”了从环境中其他部分所发来的光。相反，逐步求精方法将从一个面片来的辐射度“发射”到周围环境中。这样做的直接方法是修改式(14-41)以得到：

$$B_j\text{归因于}B_i = \rho_j B_i F_{j-i}\text{，对所有的}j \tag{14-42}$$

给定B_j的估计值，面片i对环境中其他部分的贡献可以由式(14-42)的计算决定。不幸的是，这要求对于每一个j都需要知道F_{j-i}，它由不同的半立方体决定。这使得算法具有了原有算法的庞大时空耗费。但利用式(14-34)中的相互关系，可以将式(14-42)改写为：

$$B_j\text{归因于}B_i = \rho_j B_i F_{i-j}(A_i/A_j)\text{，对所有的}j \tag{14-43}$$

对每个j用等式进行计算仅仅只需用一个以面片i为中心的半立方体计算的形状因子。如果来自面片i的形状因子能很快通过计算得到（例如，用z缓存硬件），那么只要那些面片i的辐射度计算完毕，就可以将它们丢弃，因而，在某个时间内只需要计算并存储一个半立方体和它的形状因子。

在一个面片的辐射度“辐射”出去以后，即选择另一面片进行处理。一个面片在其他面片新的光线辐射过来以后或许会被重新选择进行辐射，因而面片i“发射”的并不是它总的辐射度，而仅仅是ΔB_i自从上次发散后面片i所收到的辐射度。算法迭代进行直到达到预定的容差。算法不是随机地选择面片，而是选择那些将产生最大差异的面片，也就是那些具有最高待射能量的面片。既然辐射度是以单位面积计算的，则应选择$\Delta B_i A_i$最大的那个面片。初始时，对所有面片$B_i = \Delta B_i = E_i$，仅仅对光源非零。迭代的单步伪代码如程序14-2所示。

程序14-2 从一个面片“发射”辐射度的伪代码

```
选择面片i;

对每个面片j计算F_{i-j};

for（每个面片j）{
    ΔRadiosity = ρ_j ΔB_i F_{i-j} A_i/A_j;
    ΔB_j += ΔRadiosity;
    B_j += ΔRadiosity;
}

ΔB_i = 0;
```

程序14-2中伪代码的每一次执行都会使得另一面片将其未“发射”的辐射度发散到场景中。因此，首次执行以后，仅有光源或直接被首次选中向外“发射”的面 520

片照到的表面才是亮的。如果每迭代一次都绘3制一个新的画面，那么第一次产生的画面将相对较暗，接下来的画面将逐渐变亮。为了使早期产生的画面更加合理，我们可以在辐射度中加入一个泛光项。随着循环的每一步迭代，泛光项将逐渐减少，最终趋于零。彩图44是用泛光项进行绘制的，描绘了迭代1、2、24和100次所产生的图像。

14.9 绘制流水线

我们知道了可见面判定、光照和明暗处理的各种方法，现在可以考虑将这些处理步骤装入第7章介绍的标准图形流水线中。为简单起见，除特别说明外，我们均假设场景是由多边形构成的。在[FOLE90]的第18章中将更详细地讨论如何用硬件实现这些流水线。

14.9.1 局部光照绘制流水线

1. *z*缓存和Gouraud明暗处理

对标准流水线最直接的修改是用*z*缓存可见面判定算法来绘制有Gouraud明暗处理的多边形，如图14-41所示。*z*缓存算法的优点是对图元的出现顺序没有要求。因而，就可以像以前一样，通过遍历数据库得到图元并通过模型变换将其变换到世界坐标系中。

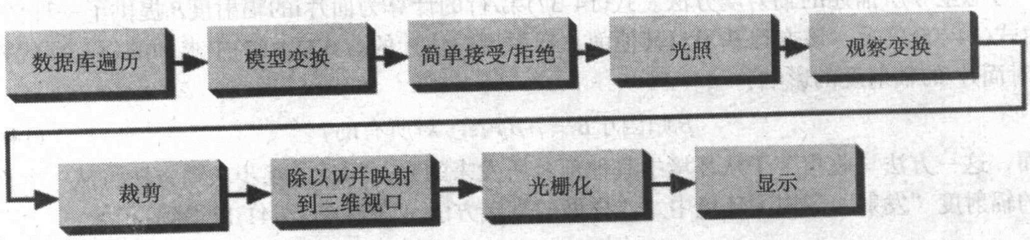


图14-41 *z*缓存和Gouraud明暗处理的绘制流水线

当建立模型时就可能为图元定义了相应的表面法向量。由于光照处理需要用到表面法向量，必须用附录中所讨论的方法将法向量进行正确的变换。我们不能试图用正确变换后的顶点来重新计算新的法向量而忽略了原来存储的法向量。和物体同时定义的法向量可能代表了其表面的真正几何属性，或者可能具体指定了用户所定义的表面混和效果，而不仅仅是由近似多边形网
521 格中相邻面的法向的平均。

下一步是要除去完全落在窗口外面的图元以及反向面剔除。现在进行这些枝节去除操作是因为我们不希望在接下来的光照处理中做无谓的工作。由于采用Gouraud明暗处理，需要在各个顶点对光照方程进行计算。这一处理必须在观察变换（包括*skew*或透视变换）前的世界坐标系（或任何与其尺寸相同的坐标系）中进行，以保持光线和表面间正确的角度和距离。如果物体没有提供顶点的法向量，则在对顶点进行光照计算前必须予以计算。剔除工作和光照处理通常是在光照坐标系下完成的，该坐标系通过对WC坐标系进行刚体变换得到的（比如，用标准的PHIGS工具创建视线方向矩阵而得到的VRC）。

接着物体通过观察变换变换到NPC，并由视见体进行裁剪。将顶点的齐次坐标除以其相应的*W*分量，并将物体映射到视口。如果一个物体部分被裁剪，那么必须为裁剪过程中产生的新顶点计算正确的亮度值。在这一刻，裁剪后的图元被交给*z*缓存算法，由其进行光栅化。在此过程中，隔行扫描转换时要对每个像点的*z*值和亮度值进行插值计算。如果确定像素是可见的，其亮度值可以进一步由深度提示效果（式(14-11)）修正，这里就不再说明。

虽然这一流水线看上去简单直观，但是必须处理许多新问题以确保高效正确的实现。例如，

考虑曲面处理所产生的问题，诸如必须进行网格化的B样条曲面片。应该在变换到屏幕大小可定的坐标系后才能进行网格化，这使得网格大小可自适应地调整，并且限制了变换的数据量。另一方面，网格化的图元必须在与世界坐标系同构的坐标系中进行光照处理。Abi-Ezzi[ABIE 89]讨论了这些问题，提出了更有效但更复杂的结合反馈循环的绘制流水线表示。这一流水线用到的光照坐标系是世界坐标系WC的等距（如刚体或欧氏）变换得到的，但计算上接近于设备坐标系（DC）以允许网格化更为有效地进行。

2. *z*缓存和Phong明暗处理

上述简单的流水线若要适应Phong明暗处理则必须加以修改，如图14-42所示。因为Phong明暗处理对表面法向量而不是亮度值进行插值，所以光照处理不能在绘制流水线的早期执行。相反，每一物体要先进行裁剪（用插值法计算出新产生顶点的法向量）、观察变换并交给*z*缓存算法处理。最后，利用在扫描转换时用插值计算得出的法向量进行光照处理。因此，每一点和其法向量必须反向映射至与世界坐标系等距的坐标系中来计算光照方程。

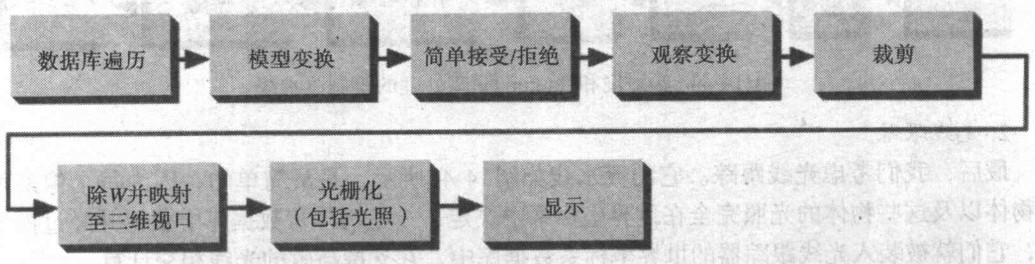


图14-42 *z*缓存和Phong明暗法的图形绘制流水线

3. 列表优先级算法和Phong明暗处理

使用列表优先级算法时，经由数据库遍历得到的并由模型变换处理后的图元被放入一个独立的数据库中，如BSP树，作为初始可见面判定的部分。图14-43描述了采用BSP树算法的流水线，其中初始可见面判定是与视点无关的。如我们第7章中所述，应用程序和图形软件包可以有各自不同的数据库。这里我们看到绘制也需要自己单独的数据库。在此情况下，多边形被分割，就必须为这些新产生的顶点确立正确的明暗信息。现在可以遍历这个绘制数据库生成从后向前排列的图元序列。建立这种数据库可以用来生成多幅图像，因而把它看成一个单独的流水线，该流水线的输出是一个新的数据库。从该数据库中取出的图形基元被裁剪、规格化，并交给该流水线的余下阶段。这些阶段的构造和*z*缓存流水线构造类似，区别在于，它们惟一需要执行的可见面判定过程是必须保证每个多边形正确地覆盖与它相交的已扫描转换的多边形。

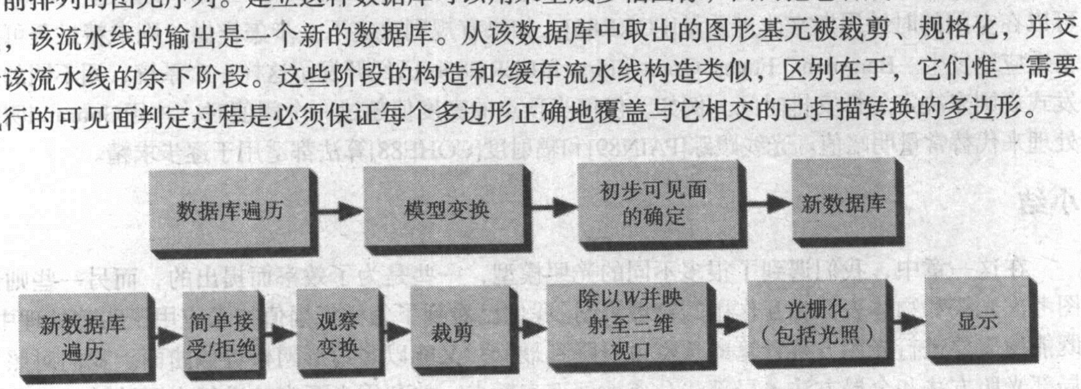


图14-43 列表优先级算法和Phong明暗处理的绘制流水线

14.9.2 全局光照绘制流水线

以上的讨论中忽略了全局光照。正如我们以前注意到的，加入全局光照效果需要考虑正在

522~523 绘制的物体与世界中其他物体的几何关系。在阴影处理中，由于阴影只依赖于光源的位置而不依赖于观察者的位置，通过对场景预处理，为物体表面加入阴影细节多边形，从而可以使用一种不同于传统绘制流水线的方法。

1. 辐射度

漫射辐射度算法为如何利用传统流水线来实现全局光照效果提供了一个很有价值的例子。14.8节的算法处理物体并且为它们赋予一系列与视点无关的顶点亮度。这些物体用修改后的z缓存和Gouraud明暗处理的流水线进行处理，如图14-44所示，该流水线不需要光照处理阶段。

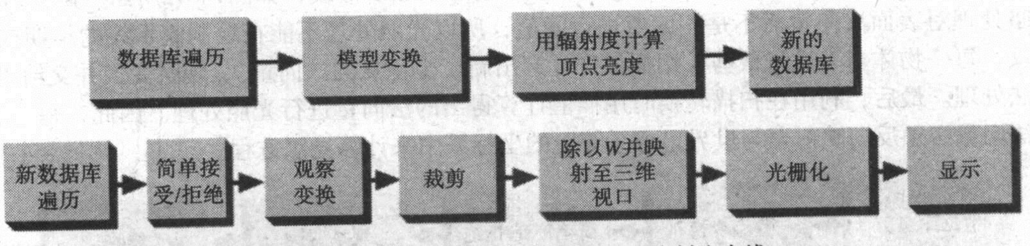


图14-44　辐射度和Gouraud明暗处理的绘制流水线

2. 光线跟踪

最后，我们考虑光线跟踪。它的流水线如图14-45所示，是最简单的，因为每个像素可见的物体以及这些物体的光照完全在世界坐标系中决定，一旦物体从数据库中获得并经过模型变换，它们就被装入光线跟踪器的世界坐标系数据库中，它支持高效的光线相交计算。

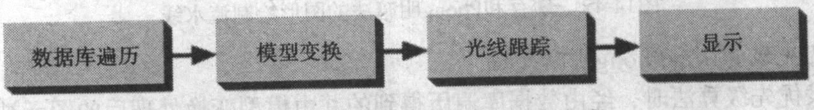


图14-45　光线跟踪的图形显示流水线

14.9.3　逐步求精方法

考虑到我们观察画面的时间有限，可以对我们所讨论的流水线做一种有价值的改善。我们不再一次就生成一幅画面的最终图像，而是首先较粗略地绘制该图形画面，然后逐步求精完善它。例如，初步的图像可以没有反走样、简单的物体模型、简单的明暗处理。当用户观察这一图像时，
524 可以在空闲的时钟周期来改进质量[FORR85]。如果有规则决定下一步怎样做，则求精过程可以自适应地发生。Bergman、Fuchs, Grant 和Spach[BERG86b]已经开发了这样一个系统，用不同的启发式规则来决定怎样安排时间。例如，仅仅当顶点的亮度值超过一个阈值时，才用Gouraud明暗处理来代替常量明暗值。光线跟踪[PAIN89]和辐射度[COHE88]算法都适用于逐步求精。

小结

在这一章中，我们遇到了很多不同的光照模型，一些是为了效率而提出的，而另一些则试图考虑光线和物体表面相互作用的物理机制。我们已看到了怎样把插值技术应用于明暗处理中，既能使需要进行光照方程计算的顶点数目降到最低，又能以多边形网格近似曲面。我们对照了局部光照方法和全局方法，局部法分离地处理表面点，并利用光源直接进行光照计算，而全局法支持环境中物体间的折射和反射。

如我们在本章一直强调的那样，不同的光照算法和明暗处理对相同的场景和相同的观察规范能产生不同的图像。选用哪种算法取决于很多因素，包括绘制图像的目的。尽管通常情况下，

为了效率而不得不牺牲真实感，但算法和硬件的改进很快会使物理上正确的全局光照模型的实时实施成为现实。然而，当效率不再成为问题时，我们仍然愿意生成一些没有纹理、阴影、反射或折射效果的图像，因为在一些情况下，这种选择仍然是传递给观察者所需信息的最好方法。

习题

14.1 (a) 在Phong光照模型中使用不同项$(\overline{N}\cdot\overline{H})^n$和$(\overline{R}\cdot\overline{V})^n$生成画面时，描述所生成的画面的不同。

(b) 证明：当图14-12中的所有向量共面时，$\alpha = 2\beta$。

(c) 证明在通常情况下这种关系并不正确。

14.2 证明：沿多边形各边和扫描线插值顶点信息的结果，其结果在三角形的情况下与方向无关。

14.3 假设存在多边形A、B、C，以与观察者距离递增的顺序与同一投影线相交。证明：如果多边形A和B透明，为它们投影的相交区域上的像素计算的颜色依赖于对式(14-23)求值时多边形A和B的计算顺序（A和B是作为多边形1和2还是作为多边形2和1）。 525

14.4 考虑使用纹理映射来修改或代替不同的材质属性。列出所有您认为可以通过单独映射属性和混合映射属性所产生的效果问题。

14.5 叙述你认为可以通过推广应用Warn的挡板和锥体概念所能模拟的物体表面其他光照效果。

14.6 根据13.4节和14.7节的内容实现简单的递归光线跟踪。

14.7 解释为什么图14-41给出的绘制流水线中的光照处理必须在裁剪之前进行。

14.8 实现一个局部光照模型的实验平台。将记有每个像素的可见多边形指针、该可见面法向、该可见面到视点的距离以及到一个或多个光源的距离和向量方向的图像存入一个材质属性表中。该平台中允许用户改变光照模型、光源的亮度和颜色以及物体表面的属性，并在每次改变时绘制画面。使用式(14-20)和光源衰减（式(14-8)）以及深度提示（式(14-11)）。 526

参考文献

下面给出的是计算机图形学方面重要的参考文献。除了各章引用的参考文献外，这里尽可能详尽地提供了便于查阅的出处。仅从书名和论文的题目也可以获知本研究领域中已经研究和正在研究的方向和思路。

因为某些期刊的引用率非常高，所以这里对期刊名做了简写。其中最重要的是ACM SIGGRAPH国际会议，其中的论文每年在《Computer Graphics》期刊上发表，另一个就是《ACM Transactions on Graphics》，这两个期刊的文献占了这里给出的总文献数量的三分之一以上。

缩写词

ACM TOG	*Association for Computing Machinery, Transactions on Graphics*
CACM	*Communications of the ACM*
CG & A	*IEEE Computer Graphics and Applications*
CGIP	*Computer Graphics and Image Processing*
CVGIP	*Computer Vision, Graphics, and Image Processing (formerly CGIP)*
FJCC	*Proceedings of the Fall Joint Computer Conference*
JACM	*Journal of the ACM*
NCC	*Proceedings of the National Computer Conference*
SJCC	*Proceedings of the Spring Joint Computer Conference*
SIGGRAPH 76	*Proceedings of SIGGRAPH '76 (Philadelphia, Pennsylvania, July 14–16, 1976)*. In *Computer Graphics*, 10(2), Summer 1976, ACM SIGGRAPH, New York.
SIGGRAPH 77	*Proceedings of SIGGRAPH '77 (San Jose, California, July 20–22, 1977)*. In *Computer Graphics*, 11(2), Summer 1977, ACM SIGGRAPH, New York.
SIGGRAPH 78	*Proceedings of SIGGRAPH '78 (Atlanta, Georgia, August 23–25, 1978)*. In *Computer Graphics*, 12(3), August 1978, ACM SIGGRAPH, New York.
SIGGRAPH 79	*Proceedings of SIGGRAPH '79 (Chicago, Illinois, August 8–10, 1979)*. In *Computer Graphics*, 13(2), August 1979, ACM SIGGRAPH, New York.
SIGGRAPH 80	*Proceedings of SIGGRAPH '80 (Seattle, Washington, July 14–18, 1980)*. In *Computer Graphics*, 14(3), July 1980, ACM SIGGRAPH, New York.
SIGGRAPH 81	*Proceedings of SIGGRAPH '81 (Dallas, Texas, August 3–7, 1981)*. In *Computer Graphics*, 15(3), August 1981, ACM SIGGRAPH, New York.
SIGGRAPH 82	*Proceedings of SIGGRAPH '82 (Boston, Massachusetts, July 26–30, 1982)*. In *Computer Graphics*, 16(3), July 1982, ACM SIGGRAPH, New York.
SIGGRAPH 83	*Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25–29, 1983)*. In *Computer Graphics*, 17(3), July 1983, ACM SIGGRAPH, New

York.

SIGGRAPH 84 *Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23–27, 1984)*. In *Computer Graphics*, 18(3), July 1984, ACM SIGGRAPH, New York.

SIGGRAPH 85 *Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26, 1985)*. In *Computer Graphics*, 19(3), July 1985, ACM SIGGRAPH, New York.

SIGGRAPH 86 *Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18–22, 1986)*. In *Computer Graphics*, 20(4), August 1986, ACM SIGGRAPH, New York.

SIGGRAPH 87 *Proceedings of SIGGRAPH '87 (Anaheim, California, July 27–31, 1987)*. In *Computer Graphics*, 21(4), July 1987, ACM SIGGRAPH, New York.

SIGGRAPH 88 *Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1–5, 1988)*. In *Computer Graphics*, 22(4), August 1988, ACM SIGGRAPH, New York.

SIGGRAPH 89 *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989)*. In *Computer Graphics*, 23(3), July 1989, ACM SIGGRAPH, New York.

ABIE89 Abi-Ezzi, S.S., *The Graphical Processing of B-Splines in a Highly Dynamic Environment*, Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, May 1989.

ADOB85 Adobe Systems, Inc., *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1985.

AMAN87 Amanatides, J., and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," in Kunii, T.L., ed., *Computer Graphics 87: Proceedings of Computer Graphics International 87, London, October 1987*, Springer, Tokyo, 1987, 149–155.

ANSI85 ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Computer Graphics—Graphical Kernel System (GKS) Functional Description*, ANSI X3.124-1985, ANSI, New York, 1985.

ANSI88 ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, Archive File Format, Clear-Text Encoding of Archive File*, ANSI, X3.144-1988, ANSI, New York, 1988.

APPE68 Appel, A., "Some Techniques for Shading Machine Renderings of Solids," *SJCC*, 1968, 37–45.

APPL85 Apple Computer, Inc., *Inside Macintosh*, Addison-Wesley, Reading, MA, 1985.

APT85 Apt, C., "Perfecting the Picture," *IEEE Spectrum*, 22(7), July 1985, 60–66.

ATHE81 Atherton, P.R., "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display," *SIGGRAPH 81*, 279–287.

ATHE83 Atherton, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," *SIGGRAPH 83*, 73–82.

BAEC87 Baecker, R., and B. Buxton, *Readings in Human-Computer Interaction*, Morgan Kaufmann, Los Altos, CA, 1987.

BALD85 Baldauf, D., "The Workhorse CRT: New Life," *IEEE Spectrum*, 22(7), July

1985, 67–73.

BANC77 Banchoff, T.F., and C.M. Strauss, *The Hypercube: Projections and Slicings*, Film, International Film Bureau, 1977.

BANC83 Banchoff, T., and J. Wermer, *Linear Algebra Through Geometry*, Springer-Verlag, New York, 1983.

BARS83 Barsky, B., and J. Beatty, "Local Control of Bias and Tension in Beta-Splines," *ACM TOG*, 2(2), April 1983, 109–134.

BARS88 Barsky, B., *Computer Graphics and Geometric Modeling Using Beta-splines*, Springer-Verlag, New York, 1988.

BART87 Bartels, R., J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA, 1987.

BASS90 Bass, C., M. Capsimalis, N. Jeske, A. Look, S, Sheppard, and G. Wiegand, *HOOPS Graphics System User Guides*, Ithaca Software, 1990.

BAUM74 Baumgart, B.G., *Geometric Modeling for Computer Vision*, Ph.D. Thesis, Report AIM-249, STAN-CS-74-463, Computer Science Department, Stanford University, Palo Alto, CA, October 1974.

BAUM75 Baumgart, B.G., "A Polyhedron Representation for Computer Vision," *NCC 75*, 589–596.

BAYE73 Bayer, B.E., "An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures," in *Conference Record of the International Conference on Communications*, 1973, 26-11–26-15.

BEAT82 Beatty, J.C., and K.S. Booth, eds., *Tutorial: Computer Graphics*, Second Edition, IEEE Comp. Soc. Press, Silver Spring, MD 1982.

BEDF58 Bedford, R., and G. Wyszecki, "Wavelength Discrimination for Point Sources," *Journal of the Optical Society of America*, 48, 1958, 129–ff.

BERG86a Bergeron, P., "A General Version of Crow's Shadow Volumes," *CG & A*, 6(9), September 1986, 17–28.

BERG86b Bergman, L., H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement," *SIGGRAPH 86*, 29–37.

BERK82 Berk, T., L. Brownston, and A. Kaufman, "A New Color-Naming System for Graphics Languages," *CG & A*, 2(3), May 1982, 37–44.

BEZI70 Bézier, P., *Emploi des Machines á Commande Numérique*, Masson et Cie, Paris, 1970. Translated by Forrest, A. R., and A. F. Pankhurst as Bézier, P., *Numerical Control—Mathematics and Applications*, Wiley, London, 1972.

BEZI74 Bézier, P., "Mathematical and Practical Possibilities of UNISURF," in Barnhill, R. E., and R. F. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, New York, 1974.

BILL81 Billmeyer, F., and M. Saltzman, *Principles of Color Technology*, second edition, Wiley, New York, 1981.

BINF71 Binford, T., in *Visual Perception by Computer, Proceedings of the IEEE Conference on Systems and Control*, Miami, FL, December 1971.

BIRR61 Birren, R., *Creative Color*, Van Nostrand Reinhold, New York, 1961.

BISH60 Bishop, A., and M. Crook, *Absolute Identification of Color for Targets Presented Against White and Colored Backgrounds*. Report WADD TR 60-611, Wright Air Development Division, Wright Patterson AFB, Dayton, Ohio, 1960.

BISH86 Bishop, G., and D.M. Weimer, "Fast Phong Shading," *SIGGRAPH 86*, 103–106.

BLIN76 Blinn, J.F., and M.E. Newell, "Texture and Reflection in Computer Generated Images," *CACM*, 19(10), October 1976, 542–547. Also in BEAT82, 456–461.

BLIN77a Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," *SIGGRAPH 77*, 192–198. Also in FREE80, 316–322.

BLIN77b Blinn, J.F., "A Homogeneous Formulation for Lines in 3-Space," *SIGGRAPH 77*, 237–241.

BLIN78a Blinn, J.F., and M.E. Newell, "Clipping Using Homogeneous Coordinates," *SIGGRAPH 78*, 245–251.

BLIN78b Blinn, J.F., "Simulation of Wrinkled Surfaces," *SIGGRAPH 78*, 286–292.

BLIN88 Blinn, J.F., "Me and My (Fake) Shadow," *CG & A*, 9(1), January 1988, 82–86.

BOUK70a Bouknight, W.J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations," *CACM*, 13(9), September 1970, 527–536. Also in FREE80, 292–301.

BOUK70b Bouknight, W.J., and K.C. Kelly, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC*, AFIPS Press, Montvale, NJ, 1970, 1–10.

BOUV85 Bouville, C., "Bounding Ellipsoids for Ray-Fractal Intersection," *SIGGRAPH 85*, 45–52.

BOYN79 Boynton, R.M., *Human Color Vision*, Holt, Rinehart, and Winston, New York, 1979.

BOYS82 Boyse, J.W., and J.E. Gilchrist, "GMSolid: Interactive Modeling for Design and Analysis of Solids," *CG & A*, 2(2), March 1982, 27–40.

BÖHM80 Böhm, W., "Inserting New Knots into B-spline Curves," *Computer Aided Design*, 12(4), July 1980, 199–201.

BÖHM84 Böhm, W., G. Farin, and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design*, 1(1), July 1984, 1–60.

BRAI78 Braid, I.C., R.C. Hillyard, and I.A. Stroud, *Stepwise Construction of Polyhedra in Geometric Modelling*, CAD Group Document No. 100, Cambridge University, Cambridge, England, 1978. Also in K.W. Brodlie, ed., *Mathematical Methods in Computer Graphics and Design*, Academic Press, New York, 1980, 123–141.

BRES65 Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal*, 4(1), 1965, 25–30.

BRES77 Bresenham, J.E. "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *Communications of the ACM*, 20(2), February 1977, 100–106.

BRES83 Bresenham, J.E., D.G. Grice, and S.C. Pi, "Bi-Directional Display of Circular Arcs," US Patent 4,371,933, February 1, 1983.

BRIT78 Britton, E., J. Lipscomb, and M. Pique, "Making Nested Rotations Convenient for the User," *SIGGRAPH 78*, 222–227.

BROT84 Brotman, L.S., and N.I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *CG & A*, 4(10), October 1984, 5–12.

BUIT75 Bui-Tuong, Phong, "Illumination for Computer Generated Pictures," *CACM*, 18(6), June 1975, 311–317. Also in BEAT82, 449–455.

CABR87 Cabral, B., N. Max, and R. Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps," *SIGGRAPH 87*, 273–281.

CACM84 "An Interview with Andries van Dam," *CACM*, 27(7), August 1984.

CARD83 Card, S., T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

CARL78 Carlbom, I., and J. Paciorek, "Planar Geometric Projections and Viewing Transformations," *Computing Surveys*, 10(4), December 1978, 465–502.

CARP84 Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method," *SIGGRAPH 84*, 103–108.

CATM74 Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

CATM75 Catmull, E., "Computer Display of Curved Surfaces," in *Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures*, May 1975. Also in FREE80, 309–315.

CHAP72 Chapanis, A., and R. Kinkade, "Design of Controls," in Van Cott, H., and R. Kinkade, eds., *Human Engineering Guide to Equipment Design*, U.S. Government Printing Office, 1972.

CHAS81 Chasen, S.H., "Historical Highlights of Interactive Computer Graphics," *Mechanical Engineering*, 103, ASME, November 1981, 32–41.

CHIN89 Chin, N., and S. Feiner, "Near Real-Time Shadow Generation Using BSP Trees," *SIGGRAPH 89*, 99–106.

CHIN90 Chin, N., *Near Real-Time Object-Precision Shadow Generation Using BSP Trees*, M.S. Thesis, Department of Computer Science, Columbia University, New York, 1990.

CHUN89 Chung, J.C., *et al.*, "Exploring Virtual Worlds with Head-Mounted Displays," *Proc. SPIE Meeting on Non-Holographic True 3-Dimensional Display Technologies*, 1083, Los Angeles, Jan. 15–20, 1989.

CLAR76 Clark, J.H., "Hierarchical Geometric Models for Visible Surface Algorithms," *CACM*, 19(10), October 1976, 547–554. Also in BEAT82, 296–303.

CLAR79 Clark, J., "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces," abstract in *SIGGRAPH 79*, 174. Also in Whitted, T., and R. Cook, eds., *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, TX, August 1986. Also in JOY88, 88–93.

CLEV83 Cleveland, W., and R. McGill, "A Color-Caused Optical Illusion on a Statistical Graph," *The American Statistician*, 37(2), May 1983, 101–105.

COHE85 Cohen, M.F., and D.P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *SIGGRAPH 85*, 31–40.

COHE88 Cohen, M.F., S.E. Chen, J.R. Wallace, and D.P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *SIGGRAPH 88*, 75–84.

CONR85 Conrac Corporation, *Raster Graphics Handbook*, second edition, Van Nostrand Reinhold, New York, 1985.

COOK82 Cook, R., and K. Torrance, "A Reflectance Model for Computer Graphics," *ACM TOG*, 1(1), January 1982, 7–24.

COOK84 Cook, R.L., "Shade Trees," *SIGGRAPH 84*, 223–231.

COWA83 Cowan, W., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates," *SIGGRAPH 83*, 315–321.

CROW77 Crow, F.C., "Shadow Algorithms for Computer Graphics," *SIGGRAPH 77*, 242–247. Also in BEAT82, 442–448.

CROW87 Crow, F.C., "The Origins of the Teapot," *CG & A*, 7(1), January 1987, 8–19.

CYRU78 Cyrus, M. and J. Beck, "Generalized Two- and Three-Dimensional Clipping," *Computers and Graphics*, 3(1), 1978, 23–28.

DEBO78 de Boor, C., *A Practical Guide to Splines*, Applied Mathematical Sciences Volume 27, Springer-Verlag, New York, 1978.

DIPP84 Dippé, M., and J. Swensen, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *SIGGRAPH 84*, 149–158.

DUFF79 Duff, T., "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays," *SIGGRAPH 79*, 270–275.

DURB88 Durbeck, R., and S. Sherr, eds., *Output Hardcopy Devices*, Academic Press, New York, 1988.

DUVA90 Duvanenko, V., W.E. Robbins, and R.S. Gyurcsik, "Improved Line Segment Clipping," *Dr. Dobb's Journal*, July 1990, 36–45, 98–100.

DVOR43 Dvořák, A., "There Is a Better Typewriter Keyboard," *National Business Education Quarterly*, 12(2), 1943, 51–58.

ENCA72 Encarnação, J., and W. Giloi, "PRADIS—An Advanced Programming System for 3-D-Display," *SJCC*, AFIPS Press, Montvale, NJ, 1972, 985–998.

ENGE68 Englebart, D.C., and W.K. English, *A Research Center for Augmenting Human Intellect*, *FJCC* , Thompson Books, Washington, D.C., 1968, 395.

FAUX79 Faux, I.D., and M.J. Pratt, *Computational Geometry for Design and Manufacture*, Wiley, New York, 1979.

FITT54 Fitts, P., "The Information Capacity of the Human Motor System in Controlling Amplitude of Motion," *Journal of Experimental Psychology*, 47(6), June 1954, 381–391.

FIUM89 Fiume, E.L., *The Mathematical Structure of Raster Graphics*, Academic Press, San Diego, 1989.

FLOY75 Floyd, R., and Steinberg, L., "An Adaptive Algorithm for Spatial Gray Scale," in *Society for Information Display 1975 Symposium Digest of Technical Papers*, 1975, 36.

FOLE82 Foley, J., and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

FOLE84 Foley, J., V. Wallace, and P. Chan, "The Human Factors of Computer Graphics Interaction Techniques," *CG & A*, 4(11), November 1984, 13–48.

FOLE87 Foley, J., "Interfaces for Advanced Computing," *Scientific American*, 257(4), October 1987, 126–135.

FOLE90 Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice, Second Edition*, Addison-Wesley, Reading, MA, 1990.

FORR79 Forrest, A.R., "On the Rendering of Surfaces," *SIGGRAPH 79*, 253–259.

FORR80 Forrest, A.R., "The Twisted Cubic Curve: A Computer-Aided Geometric Design Approach," *Computer Aided Design*, 12(4), July 1980, 165–172.

FORR85 Forrest, A.R., "Antialiasing in Practice," in Earnshaw, R.A., ed., *Fundamental Algorithms for Computer Graphics*, NATO ASI Series F: Computer and Systems Sciences, Vol. 17, Springer-Verlag, New York, 1985, 113–134.

FOUR82 Fournier, A., D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models," *CACM*, 25(6), June 1982, 371–384.

FOUR88 Fournier, A. and D. Fussell, "On the Power of the Frame Buffer," *ACM TOG*, 7(2), April 1988, 103–128.

FRAN80 Franklin, W.R., "A Linear Time Exact Hidden Surface Algorithm," *SIGGRAPH 80*, 117–123.

FRAN81 Franklin, W.R., "An Exact Hidden Sphere Algorithm that Operates in Linear Time," *CGIP,* 15(4), April 1981, 364–379.

FREE80 Freeman, H. ed., *Tutorial and Selected Readings in Interactive Computer Graphics,* IEEE Comp. Soc. Press, Silver Spring, MD 1980.

FROM84 Fromme, F., "Improving Color CAD Systems for Users: Some Suggestions from Human Factors Studies," *IEEE Design and Test of Computers,* 1(1), February 1984, 18–27.

FUCH80 Fuchs, H., Z.M. Kedem, and B.F. Naylor, "On Visible Surface Generation by A Priori Tree Structures," *SIGGRAPH 80,* 124–133.

FUCH83 Fuchs, H., G.D. Abram, and E.D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *SIGGRAPH 83,* 65–72.

FUJI85 Fujimura, K., and Kunii, T. L., "A Hierarchical Space Indexing Method," in Kunii, T.L., ed., *Computer Graphics: Visual Technology and Art, Proceedings of Computer Graphics Tokyo '85 Conference,* Springer-Verlag, 1985, 21–34.

GILO78 Giloi, W.K., *Interactive Computer Graphics—Data Structures, Algorithms, Languages,* Prentice-Hall, Englewood Cliffs, NJ, 1978.

GLAS84 Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *CG & A,* 4(10), October 1984, 15–22.

GLAS89 Glassner, A.S., ed., *An Introduction to Ray Tracing,* Academic Press, London, 1989.

GOLD71 Goldstein, R.A., and R. Nagel, "3-D Visual Simulation," *Simulation,* 16(1), January 1971, 25–31.

GOLD83 Goldberg, A., and D. Robson, *SmallTalk 80: The Language and Its Implementation,* Addison-Wesley, Reading, MA, 1983.

GORA84 Goral, C.M., K.E. Torrance, D.P. Greenberg, and B. Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *SIGGRAPH 84,* 213–222.

GOSS88 Gossard, D., R. Zuffante, and H. Sakurai, "Representing Dimensions, Tolerances, and Features in MCAE Systems," *CG & A,* 8(2), March 1988, 51–59.

GOUR71 Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. on Computers,* C-20(6), June 1971, 623–629. Also in FREE80, 302–308.

GREE87 Greenstein, J. and L. Arnaut, "Human Factors Aspects of Manual Computer Input Devices," in Salvendy, G., ed., *Handbook of Human Factors,* Wiley, New York, 1987, 1450–1489.

GREG66 Gregory, R.L., *Eye and Brain— The Psychology of Seeing,* McGraw-Hill, New York, 1966.

GREG70 Gregory, R.L., *The Intelligent Eye,* McGraw-Hill, London, 1970.

GSPC77 Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee," *Computer Graphics,* 11, 1977.

GSPC79 Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee," *Computer Graphics,* 13(3), August 1979.

HAEU76 Haeusing, M., "Color Coding of Information on Electronic Displays," in *Proceedings of the Sixth Congress of the International Ergonomics Association,* 1976, 210–217.

HAGE86 Hagen, M., *Varieties of Realism,* Cambridge University Press, Cambridge, England, 1986.

HAIN89 Haines, E., "Essential Ray Tracing Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing,* Academic Press, London, 1989, 33–77.

HALL89 Hall, R., *Illumination and Color in Computer Generated Imagery,* Springer-Verlag, New York, 1989.

HAML77 Hamlin, G., Jr., and C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques," *SIGGRAPH 77,* 206–213. Also in FREE80, 264–271.

HANR83 Hanrahan, P., "Ray Tracing Algebraic Surfaces," *SIGGRAPH 83,* 83–90.

HANR89 Hanrahan, P., "A Survey of Ray-Surface Intersection Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing,* Academic Press, London, 1989, 79–119.

HE92 He, X., P. Heynen, R. Phillips, K. Torrance, D. Salesin, and D. Greenberg, "A Fast and Accurate Light Reflection Model," *SIGGRAPH 92,* 253–254.

HECK84 Heckbert, P.S., and P. Hanrahan, "Beam Tracing Polygonal Objects," *SIGGRAPH 84,* 119–127.

HECK86a Heckbert, P.S., "Filtering by Repeated Integration," *SIGGRAPH 86,* 315–321.

HECK86b Heckbert, P.S., "Survey of Texture Mapping," *CG & A,* 6(11), November 1986, 56–67.

HEDG82 Hedgley, D.R., Jr., *A General Solution to the Hidden-Line Problem,* NASA Reference Publication 1085, NASA Scientific and Technical Information Branch, 1982.

HERO76 Herot, C., "Graphical Input Through Machine Recognition of Sketches," *SIGGRAPH 76,* 97–102.

HIRS70 Hirsch, R., "Effects of Standard vs. Alphabetical Keyboard Formats on Typing Performance," *Journal of Applied Psychology,* 54, December 1970, 484–490.

HODG85 Hodges, L., and D. McAllister, "Stereo and Alternating-Pair Techniques for Display of Computer-Generated Images," *CG & A,* 5(9), September 1985, 38–45.

HOFF61 Hoffman, K., and R. Kunze, *Linear Algebra,* Prentice–Hall, Englewood Cliffs, NJ, 1961.

HUGH89 Hughes, J., *Integer and Floating-Point Z-Buffer Resolution,* Department of Computer Science Technical Report, Brown University, Providence, RI, 1989.

HUNT78 Hunter, G.M., *Efficient Computation and Data Structures for Graphics,* Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

HUNT79 Hunter, G.M. and K. Steiglitz, "Operations on Images Using Quad Trees," *IEEE Trans. Pattern Anal. Mach. Intell.,* 1(2), April 1979, 145–153.

HUNT87 Hunt, R.W., *The Reproduction of Colour,* fourth edition, Fountain Press, Tolworth, England, 1987.

HUTC86 Hutchins, E., J. Hollan, and D. Norman, "Direct Manipulation Interfaces," in Norman, D., and S. Draper, eds., *User Centered System Design,* Erlbaum, Hillsdale, NJ, 1986, 87–124.

IES87 Illuminating Engineering Society, Nomenclature Committee, *ANSI/IES RP-16-1986: American National Standard: Nomenclature and Definitions for Illuminating Engineering,* Illuminating Engineering Society of North America, New York, 1987.

INGA81 Ingalls, D., "The SmallTalk Graphics Kernel," *BYTE,* 6(8), August 1981.

INTE85 Interaction Systems, Inc., *TK-1000 Touch System,* Interaction Systems, Inc., Newtonville, MA, 1985.

INTE88 International Standards Organization, *International Standard Information Processing Systems*—Computer Graphics—Graphical Kernel System for Three Dimensions (GKS–3D) Functional Description, ISO Document Number 8805:1988(E), American National Standards Institute, New York, 1988.

JACK64 Jacks, E., "A Laboratory for the Study of Man–Machine Communication," in *FJCC 64*, AFIPS, Montvale, NJ, 1964, 343–350.

JACK80 Jackins, C., and S.L. Tanimoto, "Oct-Trees and Their Use in Representing Three-Dimensional Objects," *CGIP*, 14(3), November 1980, 249–270.

JARV76a Jarvis, J.F., C.N. Judice, and W.H. Ninke, "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays," *CGIP*, 5(1), March 1976, 13–40.

JARV76b Jarvis, J.F., and C.S. Roberts, "A New Technique for Displaying Continuous Tone Images on a Bilevel Display," *IEEE Trans.*, COMM-24(8), August 1976, 891–898.

JOBL78 Joblove, G.H., and D. Greenberg, "Color Spaces for Computer Graphics," *SIGGRAPH 78*, 20–27.

JOY86 Joy, K.I., and M.N. Bhetanabhotla, "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence," *SIGGRAPH 86*, 279–285.

JOY88 Joy, K., C. Grant, N. Max, and L. Hatfield, *Tutorial: Computer Graphics: Image Synthesis*, IEEE Computer Society, Washington, DC, 1988.

JUDD75 Judd, D., and G. Wyszecki, *Color in Business, Science, and Industry*, Wiley, New York, 1975.

JUDI74 Judice, J.N., J.F. Jarvis, and W. Ninke, "Using Ordered Dither to Display Continuous Tone Pictures on an AC Plasma Panel," *Proceedings of the Society for Information Display*, Q4 1974, 161–169.

KAJI82 Kajiya, J.T., "Ray Tracing Parametric Patches," *SIGGRAPH 82*, 245–254.

KAJI83 Kajiya, J., "New Techniques for Ray Tracing Procedurally Defined Objects," *SIGGRAPH 83*, 91–102.

KAJI85 Kajiya, J.T., "Anisotropic Reflection Models," *SIGGRAPH 85*, 15–21.

KAJI86 Kajiya, J.T., "The Rendering Equation," *SIGGRAPH 86*, 143–150.

KAPP85 Kappel, M.R., "An Ellipse-Drawing Algorithm for Raster Displays," in Earnshaw, R., ed., *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Springer-Verlag, Berlin, 1985, 257–280.

KAY79a Kay, D.S., *Transparency, Refraction and Ray Tracing for Computer Synthesized Images*, M.S. Thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, January 1979.

KAY79b Kay, D.S., and D. Greenberg, "Transparency for Computer Synthesized Images," *SIGGRAPH 79*, 158–164.

KAY86 Kay, T.L., and J.T. Kajiya, "Ray Tracing Complex Scenes," *SIGGRAPH 86*, 269–278.

KELL76 Kelly, K., and D. Judd, *COLOR—Universal Language and Dictionary of Names*, National Bureau of Standards Spec. Publ. 440, 003-003-01705-1, U.S. Government Printing Office, Washington, DC, 1976.

KLIN71 Klinger, A., "Patterns and Search Statistics," in Rustagi, J., ed., *Optimizing Methods in Statistics*, Academic Press, New York, 1971, 303–337.

KNOW77 Knowlton, K., and L. Cherry, "ATOMS—A Three-D Opaque Molecule System for Color Pictures of Space-Filling or Ball-and-Stick Models," *Computers and Chemistry*, 1, 1977, 161–166.

KNUT87 Knuth, D., "Digital Halftones by Dot Diffusion," *ACM TOG*, 6(4), October 1987, 245–273.

KORE83 Korein, J., and N. Badler, "Temporal Anti-Aliasing in Computer Generated Animation," *SIGGRAPH 83*, 377–388.

KREB79 Krebs, M., and J. Wolf, "Design Principles for the Use of Color in Displays," *Proceedings of the Society for Information Display*, 20, 1979, 10–15.

KRUE83 Krueger, M., *Artificial Reality*, Addison-Wesley, Reading, MA, 1983.

LAID86 Laidlaw, D.H., W.B. Trumbore, and J.F. Hughes, "Constructive Solid Geometry for Polyhedral Objects," *SIGGRAPH 86*, 161–170.

LAND85 Landauer, T., and D. Nachbar, "Selection from Alphabetic and Numeric Menu Trees Using a Touch-Sensitive Screen: Breadth, Depth, and Width," in *Proceedings CHI '85 Human Factors in Computing Systems Conference*, ACM, New York, 1985, 73–78.

LANE80 Lane, J., L. Carpenter, T. Whitted, and J. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *CACM*, 23(1), January 1980, 23–34. Also in BEAT82, 468–479.

LASS87 Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation," *SIGGRAPH 87*, 35–44.

LAYB79 Laybourne, K., *The Animation Book*, Crown, New York, 1979.

LEAF74 Leaf, C., *The Owl Who Married a Goose*, film, National Film Board of Canada, 1974.

LEAF77 Leaf, C., *The Metamorphosis of Mr. Samsa*, film, National Film Board of Canada, 1977.

LEVI76 Levin, J., "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces," *CACM*, 19(10), October 1976, 555–563.

LIAN84 Liang, Y-D., and Barsky, B., "A New Concept and Method for Line Clipping," *ACM TOG*, 3(1), January 1984, 1–22.

LIND68 Lindenmayer, A, "Mathematical Models for Cellular Interactions in Development, Parts I and II," *J. Theor. Biol.*, 18, 1968, 280–315.

LINT89 Linton, M., J. Vlissides, and P. Calder, "Composing User Interfaces with InterViews," *IEEE Computer*, 22(2), February 1989, 8–22.

MACH78 Machover, C., "A Brief Personal History of Computer Graphics," *Computer*, 11(11), November 1978, 38–45.

MAGI68 Mathematical Applications Group, Inc., "3-D Simulated Graphics Offered by Service Bureau," *Datamation*, 13(1), February 1968, 69.

MAHL72 Mahl, R., "Visible Surface Algorithms for Quadric Patches," *IEEE Trans. on Computers*, C-21(1), January 1972, 1–4.

MAHN73 Mahnkopf, P., and J.L. Encarnação, *FLAVIS—A Hidden Line Algorithm for Displaying Spatial Constructs Given by Point Sets*, Technischer Bericht Nr. 148, Heinrich Hertz Institut, Berlin, 1973.

MAMM89 Mammen, A., "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," *CG & A*, 9(4), July 1989, 43–55.

MAND82 Mandelbrot, B., Technical Correspondence, *CACM*, 25(8), August 1982, 581–583.

MÄNT88 Mäntylä, M. *Introduction to Solid Modeling*, Computer Science Press, Rockville, MD, 1988.

MARC82 Marcus, A., "Color: A Tool for Computer Graphics Communication," in Greenberg, D., A. Marcus, A. Schmidt, and V. Gorter, *The Computer Image*,

Addison-Wesley, Reading, MA, 1982, 76–90.

MARK80 Markowsky, G., and M.A. Wesley, "Fleshing Out Wire Frames," *IBM Journal of Research and Development*, 24(5), September 1980, 582–597.

MARS85 Marsden, J., and A. Weinstein, *Calculus I, II, and III*, second edition, Springer Verlag, New York, 1985.

MAX79 Max, N.L., "ATOMLLL: ATOMS with Shading and Highlights," *SIGGRAPH 79*, 165–173.

MAX84 Max, N.L., "Atoms with Transparency and Shadows," *CVGIP*, 27(1), July 1984, 46–63.

MAXW46 Maxwell, E.A., *Methods of Plane Projective Geometry Based on the Use of General Homogeneous Coordinates*, Cambridge University Press, Cambridge, England, 1946.

MAXW51 Maxwell, E.A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, Cambridge, England, 1951.

MAYH90 Mayhew, D., *Principles and Guidelines in User Interface Design*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

MEAG80 Meagher, D., *Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer*, Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, October 1980.

MEAG82 Meagher, D., "Geometric Modeling Using Octree Encoding," *CGIP*, 19(2), June 1982, 129–147.

MEIE88 Meier, B., "ACE: A Color Expert System for User Interface Design," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 117–128, 1988.

MEYE80 Meyer, G.W., and D.P. Greenberg, "Perceptual Color Spaces for Computer Graphics," *SIGGRAPH 80*, 254–261.

MEYE88 Meyer, G., and Greenberg, D., "Color-defective Vision and Computer Graphic Displays," *CG & A*, 8(5), September 1988, 28–40.

MICH71 Michaels, S., "QWERTY Versus Alphabetical Keyboards as a Function of Typing Skill," *Human Factors*, 13(5), October 1971, 419–426.

MICR89 Microsoft Corporation, *Presentation Manager*, Microsoft Corporation, Bellevue, WA, 1989.

MILL87 Miller, J.R., "Geometric Approaches to Nonplanar Quadric Surface Intersection Curves," *ACM TOG*, 6(4), October 1987, 274–307.

MILL89 Miller, J.R., "Architectural Issues in Solid Modelers," *CG & A*, 9(5), September 1989, 72–87.

MORT85 Mortenson, M., *Geometric Modeling*, Wiley, New York, 1985.

MUNS76 Munsell Color Company, *Book of Color*, Munsell Color Company, Baltimore, MD, 1976.

MURC85 Murch, G., "Using Color Effectively: Designing to Human Specifications," *Technical Communications*, Q4 1985, Tektronix Corporation, Beaverton, OR, 14–20.

MUSG89 Musgrave, F.K., "Prisms and Rainbows: A Dispersion Model for Computer Graphics," in *Proceedings of Graphics Interface '89*, London, Ontario, June 19–23, 1989, 227–234.

MYER68 Myer, T., and I. Sutherland, "On the Design of Display Processors," *CACM*, 11(6), June 1968, 410–414.

MYER75 Myers, A.J., *An Efficient Visible Surface Program*, Report to the National

Science Foundation, Computer Graphics Research Group, Ohio State University, Columbus, OH, July 1975.

NAYL90 Naylor, B.F., "Binary Space Partitioning Trees as an Alternative Representation of Polytopes," *CAD*, 22(4), May 1990, 250–253.

NEID93 Neider, J., T. Davis, and M. Woo, *OpenGL Programming Guide*, Addison-Wesley, Reading, MA, 1993.

NEWE72 Newell, M.E., R.G. Newell, and T.L. Sancha, "A Solution to the Hidden Surface Problem," in *Proceedings of the ACM National Conference 1972*, 443–450. Also in FREE80, 236–243.

NICH87 Nicholl, T.M., D.T. Lee, and R.A. Nicholl, "An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis," *SIGGRAPH 87*, 253–262.

NICO77 Nicodemus, F.E., J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis, *Geometrical Considerations and Nomenclature for Reflectance*, NBS Monograph 160, U.S. Department of Commerce, Washington DC, October 1977.

NISH85 Nishita, T., and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *SIGGRAPH 85*, 23–30.

NOLL67 Noll, M., "A Computer Technique for Displaying N-dimensional Hyperobjects," *CACM*, 10(8), August 1967, 469–473.

NORM88 Norman, D., *The Psychology of Everyday Things*, Basic Books, New York, 1988.

OPEN89 Open Software Foundation, *OSF/MOTIF™ Manual*, Open Software Foundation, Cambridge, MA, 1989.

OSTW31 Ostwald, W., *Colour Science*, Winsor & Winsor, London, 1931.

PAIN89 Painter, J., and K. Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *SIGGRAPH 89*, 281–288.

PALA88 Palay, A., W. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, and T. Peters, "The Andrew Toolkit: An Overview," in *Proceedings 1988 Winter USENIX*, February 1988, 9–21.

PANT91 Pantone, Inc., *PANTONE Color Formula Guide 1000*, © Pantone, Inc., 1991.

PEAC85 Peachey, D.R., "Solid Texturing of Complex Surfaces," *SIGGRAPH 85*, 279–286.

PEIT86 Peitgen, H.-O., and P.H. Richter, *The Beauty of Fractals: Images of Complex Dynamical Systems*, Springer-Verlag, Berlin, 1986.

PERL85 Perlin, K., "An Image Synthesizer," *SIGGRAPH 85*, 287–296.

PERR85 Perry, T., and P. Wallach, "Computer Displays: New Choices, New Tradeoffs," *IEEE Spectrum*, 22(7), July 1985, 52–59.

PHIG88 PHIGS+ Committee, Andries van Dam, chair, "PHIGS+ Functional Description, Revision 3.0," *Computer Graphics*, 22(3), July 1988, 125–218.

PHIG92 Programmer's Hierarchical Interactive Graphics System (PHIGS) Part 4–Plus Lumierè und Surfaces (PHIGS PLUS), ISO/IEC 9592-4:1992(E).

PHIL91 Phillips, R.L., "MediaView: A General Multimedia Digital Publishing System," *CACM*, 34(7), July 1991, 75–83.

PHIL92 Phillips, R.L., "Opportunities for Multimedia in Education," in S. Cunningham and R. Hubbold, *Interactive Learning Through Visualization*, Springer-Verlag, Berlin, 1992, 25–35.

PITT67 Pitteway, M.L.V., "Algorithm for Drawing Ellipses or Hyperbolae with a

Digital Plotter," *Computer J.*, 10(3), November 1967, 282–289.

PIXA86 Pixar Corporation, *Luxo, Jr.*, film, Pixar Corporation, San Rafael, CA, 1986.

PIXA88 Pixar Corporation, *The RenderMan Interface*, Version 3.0, Pixar Corporation, San Rafael, CA, May 1988.

PORT79 Porter, T., "The Shaded Surface Display of Large Molecules," *SIGGRAPH 79*, 234–236.

POTM82 Potmesil, M., and I. Chakravarty, "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM TOG*, 1(2), April 1982, 85–108.

PRAT84 Pratt, M., "Solid Modeling and the Interface Between Design and Manufacture," *CG & A*, 4(7), July 1984, 52–59.

PREP85 Preparata, F. P., and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

PRES88 Press, W.H., B.P. Flannery, S.A. Teukolskym, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1988.

PRIN71 Prince, D., *Interactive Graphics for Computer Aided Design*, Addison-Wesley, Reading, MA, 1971.

PRIT77 Pritchard, D.H., "U.S. Color Television Fundamentals—A Review," *IEEE Transactions on Consumer Electronics*, CE-23(4), November 1977, 467–478.

PRUS88 Prusinkiewicz, P., A. Lindenmayer, and J. Hanan, "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," *SIGGRAPH 88*, 141–150.

PUTN86 Putnam, L.K., and P.A. Subrahmanyam, "Boolean Operations on *n*-Dimensional Objects," *CG & A*, 6(6), June 1986, 43–51.

RATL72 Ratliff, F., "Contour and Contrast," *Scientific American*, 226(6), June 1972, 91–101. Also in BEAT82, 364–375.

REDD78 Reddy, D., and S. Rubin, *Representation of Three-Dimensional Objects*, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

REFF88 de Reffye, P., C. Edelin, J. Françon, M. Jaeger, and C. Puech, "Plant Models Faithful to Botanical Structure and Development," *SIGGRAPH 88*, 151–158.

REQU77 Requicha, A.A.G., *Mathematical Models of Rigid Solids*, Tech. Memo 28, Production Automation Project, University of Rochester, Rochester, NY, 1977.

REQU80 Requicha, A.A.G., "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, 12(4), December 1980, 437–464.

REQU84 Requicha, A.A.G., "Representation of Tolerances in Solid Modeling: Issues and Alternative Approaches," in Pickett, M., and J. Boyse, eds., *Solid Modeling by Computers*, Plenum Press, New York, 1984, 3–22.

REQU85 Requicha, A.A.G., and H.B. Voelcker, "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE*, 73(1), January 1985, 30–44.

ROBE65 Roberts, L.G., *Homogeneous Matrix Representations and Manipulation of N-Dimensional Constructs*, Document MS 1405, Lincoln Laboratory, MIT, Cambridge, MA, 1965.

ROMN69 Romney, G.W., G.S. Watkins, and D.C. Evans, "Real Time Display of Computer Generated Half-Tone Perspective Pictures," in *Proceedings 1968 IFIP Congress*, North Holland Publishing Co., 1969, 973–978.

ROSE85 Rose, C., B. Hacker, R. Anders, K. Wittney, M. Metzler, S. Chernicoff, C. Espinosa, A. Averill, B. Davis, and B. Howard, *Inside Macintosh*, I, Addison-Wesley, Reading, MA, 1985, I-35–I-213.

ROSS86 Rossignac, J.R., and A.A.G. Requicha, "Depth-Buffering Display Techniques for Constructive Solid Geometry," *CG & A*, 6(9), September 1986, 29–39.

RUBE84 Rubenstein, R., and H. Hersh, *The Human Factor—Designing Computer Systems for People*, Digital Press, Burlington, MA, 1984.

RUBI80 Rubin, S.M., and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *SIGGRAPH 80*, 110–116.

SALV87 Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987.

SAME84 Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Comp. Surv.*, 16(2), June 1984, 187–260.

SAME89a Samet, H., "Neighbor Finding in Images Represented by Octrees," *CVGIP*, 46(3), June 1989, 367–386.

SAME89b Samet, H. and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications," *CG & A*, 8(4), July 1988, 59–75.

SAME90a Samet, H., *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

SAME90b Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1990.

SARR83 Sarraga, R.F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID," *CVGIP*, 22(2), May 1983, 222–238.

SCHE88 Scheifler, R.W., J. Gettys, and R. Newman, *X Window System*, Digital Press, 1988.

SCHM86 Schmucker, K., "MacApp: An Application Framework," *Byte*, 11(8), August 1986, 189–193.

SCHN90 Schneider, P., "An Algorithm for Automatically Fitting Digitized Curves," *Graphics Gems*, Ed: A. Glassner, Academic Press, 1990, 612–626.

SCHU69 Schumacker, R., B. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-Generated Images to Visual Simulation*, Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX, September 1969.

SCHW82 Schweitzer, D., and E. Cobb, "Scanline Rendering of Parametric Surfaces," *SIGGRAPH 82*, 265–271.

SEDE84 Sederberg, T.W., and D.C. Anderson, "Ray Tracing of Steiner Patches," *SIGGRAPH 84*, 159–164.

SEQU89 Séquin, C.H., and E.K. Smyrl, "Parameterized Ray Tracing," *SIGGRAPH 89*, 307–314.

SHER93 Sherr, S., *Electronic Displays, Second Edition*, Wiley, New York, 1993.

SHNE86 Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1986.

SIEG81 Siegel, R., and J. Howell, *Thermal Radiation Heat Transfer*, second edition, Hemisphere, Washington, DC, 1981.

SMIT78 Smith, A.R., "Color Gamut Transform Pairs," *SIGGRAPH 78*, 12–19.

SMIT79 Smith, A.R., "Tint Fill," *SIGGRAPH 79*, 276–283.

SMIT84 Smith, A.R., "Plants, Fractals and Formal Languages," *SIGGRAPH 84*, 1–10.

SNOW83 Snowberry, K., S. Parkinson, and N. Sisson, "Computer Display Menus,"

Ergonomics, 26(7), July 1983, 699–712.

SPAR78 Sparrow, E.M., and R.D. Cess, *Radiation Heat Transfer*, Hemisphere, Washington, DC, 1978.

SPRO82 Sproull, R.F., "Using Program Transformations to Derive Line-Drawing Algorithms," *ACM TOG*, 1(4), October 1982, 259–273.

STAU78 Staudhammer, J., "On Display of Space Filling Atomic Models in Real Time," *SIGGRAPH 78*, 167–172.

STON88 Stone, M., W. Cowan, and J. Beatty, "Color Gamut Mapping and the Printing of Digital Color Images," *ACM TOG*, 7(3), October 1988, 249–292.

STOV82 Stover, H., "True Three-Dimensional Display of Computer Data," in *Proceedings of SPIE*, 367, August 1982, 141–144.

STRO91 Stroustrup, B., *The C++ Programming Language*, second edition, Addison-Wesley, Reading, MA, 1991.

SUN89 Sun Microsystems, *OPEN LOOK Graphical User Interface*, Sun Microsystems, Mountain View, CA, 1989.

SUTH63 Sutherland, I.E., "Sketchpad: A Man–Machine Graphical Communication System," in *SJCC*, Spartan Books, Baltimore, MD, 1963.

SUTH74a Sutherland, I.E., R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *ACM Computing Surveys*, 6(1), March 1974, 1–55. Also in BEAT82, 387–441.

SUTH74b Sutherland, I.E., and Hodgman, G.W., "Reentrant Polygon Clipping," *CACM*, 17(1), January 1974, 32–42.

TAMM82 Tamminen, M., and R. Sulonen, "The EXCELL Method for Efficient Geometric Access to Data," in *Proc. 19th ACM IEEE Design Automation Conf.*, Las Vegas, June 14–16, 1982, 345–351.

TANN85 Tannas, L. Jr., ed., *Flat-Panel Displays and CRTs*, Van Nostrand Reinhold, New York, 1985.

THIB87 Thibault, W.C., and B.F. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *SIGGRAPH 87*, 153–162.

THOM86 Thomas, S.W., "Dispersive Refraction in Ray Tracing," *The Visual Computer*, 2(1), January 1986, 3–8.

TILL83 Tiller, W., "Rational B-Splines for Curve and Surface Representation," *CG & A*, 3(6), September 1983, 61–69.

TILO80 Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," *IEEE Trans. on Computers*, C-29(10), October 1980, 847–883.

TORR66 Torrance, K.E., E.M. Sparrow, and R.C. Birkebak, "Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces," *J. Opt. Soc. Am.*, 56(7), July 1966, 916–925.

TORR67 Torrance, K., and E.M. Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces," *J. Opt. Soc. Am.*, 57(9), September 1967, 1105–1114.

TOTH85 Toth, D.L., "On Ray Tracing Parametric Surfaces," *SIGGRAPH 85*, 171–179.

TURN84 Turner, J.A., *A Set-Operation Algorithm for Two and Three-Dimensional Geometric Objects*, Architecture and Planning Research Laboratory, College of Architecture, University of Michigan, Ann Arbor, MI, August 1984.

ULIC87 Ulichney, R., *Digital Halftoning*, MIT Press, Cambridge, MA, 1987.

VANA84 Van Aken, J. R., "An Efficient Ellipse-Drawing Algorithm," *CG & A*, 4(9),

September 1984, 24–35.

VANA85 Van Aken, J.R., and M. Novak, "Curve-Drawing Algorithms for Raster Displays," *ACM TOG,* 4(2), April 1985, 147–169.

VOSS87 Voss, R., "Fractals in Nature: Characterization, Measurement, and Simulation," in *Course Notes 15 for SIGGRAPH 87,* Anaheim, CA, July 1987.

WACO93 Wacom Technology Corporation, *Wacom Cordless Digitizer Specification Sheet,* Wacom Technology Corporation, Vancouver, WA, 1993.

WARE87 Ware, C., and J. Mikaelian, "An Evaluation of an Eye Tracker as a Device for Computer Input," in *Proceedings of CHI + GI 1987,* ACM, New York, 183–188.

WARN69 Warnock, J., *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures,* Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, UT, June 1969.

WARN83 Warn, D.R., "Lighting Controls for Synthetic Images," *SIGGRAPH 83,* 13–21.

WATK70 Watkins, G.S., *A Real Time Visible Surface Algorithm,* Ph.D. Thesis, Technical Report UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, University of Utah, Salt Lake City, UT, June 1970.

WEGH84 Weghorst, H., G. Hooper, and D.P. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM TOG,* 3(1), January 1984, 52–69.

WEIL77 Weiler, K., and P. Atherton, "Hidden Surface Removal Using Polygon Area Sorting," *SIGGRAPH 77,* 214–222.

WEIS66 Weiss, R.A., "BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces," *JACM,* 13(2), April 1966, 194–204. Also in FREE80, 203–213.

WESL81 Wesley, M.A., and G. Markowsky, "Fleshing Out Projections," *IBM Journal of Research and Development,* 25(6), November 1981, 934–954.

WHIT80 Whitted, T., "An Improved Illumination Model for Shaded Display," *CACM,* 23(6), June 1980, 343–349.

WHIT84 Whitton, M., "Memory Design for Raster Graphics Displays," *CG & A,* 4(3), March 1984, 48–65.

WITK88 Witkin, A., and M. Kass, "Spacetime Constraints," *SIGGRAPH 88,* 159–168.

WOLF87 Wolf, C., and P. Morel-Samuels, "The Use of Hand-Drawn Gestures for Text Editing," *International Journal of Man-Machine Studies,* 27(1), July 1987, 91-102.

WOLF90 Wolff, L., and D. Kurlander, "Ray Tracing with Polarization Parameters," *CG & A,* 10(6), November 1990, 44–55.

WOLF91 Wolfram, S., *Mathematica: A System for Doing Mathematics by Computer,* second edition, Addison-Wesley, Reading, MA, 1991.

WOO85 Woo, T., "A Combinatorial Analysis of Boundary Data Structure Schemata," *CG & A,* 5(3), March 1985, 19–27.

WOON71 Woon, P.Y., and H. Freeman, "A Procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces," in *IFIP 1971,* North-Holland Pub. Co., Amsterdam, 1971, pp. 1120–1125. Also in FREE80, 230–235.

WU87 Wu, X., and J.G. Rokne, "Double-Step Incremental Generation of Lines and Circles," *CVGIP,* (37), 1987, 331-334.

WYLI67 Wylie, C., G.W. Romney, D.C. Evans, and A.C. Erdahl, "Halftone Perspective Drawings by Computer," *FJCC 67,* Thompson Books, Washington, DC, 1967, 49–58.

WYSZ82 Wyszecki, G., and W. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second edition, Wiley, New York, 1982.

WYVI88 Wyvill, B., "The Great Train Rubbery," *ACM SIGGRAPH 88 Electronic Theater and Video Review,* 26, 1988.

WYVI90 Wyvill, B., "Symmetric Double Step Line Algorithm," *Graphics Gems*, Ed. A. Glassner, Academic Press, 1990, 101–104.

ZDON90 Zdonik, S.B., and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, CA, 1990.

ZIMM87 Zimmerman, T., J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A Hand Gesture Interface Device," in *Proceedings of the CHI + GI 1987 Conference*, ACM, New York, 189–192.

索 引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

Accelerator keys（快捷键），309
ACE（彩色专家系统），419
Achromatic light（消色差光），395
ACM, 12
Active-edge table (AET)（活动边表），91, 455
Active-surface table（活动面表），458
Adaptive subdivision（自适应细分），见Spatial partitioning
Additive color primaries（附加色彩基（值）），410
Address space（地址空间），141, 151
  single（单一），150~151
Addressability（寻址能力），130
Aerial perspective（空中透视），428
Aliasing（走样），119, 430, 437, 442, 453, 另见Antialiasing
  artifact（人工痕迹），10
  sliver polygons（狭长多边形），90
  temporal（时域），435, 437
Alignment（排列、校直、准线）
  for patterns（图案），94
α (angle between $\bar{R}$ and $\bar{V}$)（$\bar{R}$和$\bar{V}$间的夹角），487
Ambient light（环境光），430, 478~479
Ambient reflection coefficient (k_a)（环境反射系数），479
American National Standards Institute（美国国家标准学会(ANSI)），见ANSI
Anchor, of pattern（图案固定），94
Animation（动画），6, 434~437
  basic rules（基本规则），436~437
  cartoon -character（卡通人物（角色）），436
  control（控制），436
  conventional（常规），435~436
  graphical languages（图形语言），435
  key-frame（关键帧），436
  staging of（分解），437
Animation control（动画控制）
  explicit（显式的），436
  key-frame（关键帧），436
ANSI (American National Standards Institute（美国国家标准学会），12, 239
Antialiasing（反走样），10, 66, 70, 79, 119, 125, 419。另见Area sampling
  temporal（时域的），435, 437
Application（应用）
  database（数据库），16
  model（模型），15~16
  program(程序), 15
Area sampling（区域采样）
  unweighted（未加权区域采样），120
  weighted（加权区域采样），122
Area subdivision algorithm, Warnock（Warnock的区域细分算法），383
Area subdivision algorithms（区域细分算法），468
Artificial reality（人造现实）。另见Virtual world
Atmospheric attenuation（大气衰减），483
Attribute bundle, PHIGS（属性包），289
Attributes（属性），27
  inheritance（继承性），273
  nongeometric（非几何），252, 275
  object（对象），16
  output primitives（输出图元），27~33
  SRGP（简单的光栅图形软件包, Simple Raster Graphics Package的缩写），27~29
Autocompletion, command（自动完成，命令），306

B

B(radiosity)（辐射度），515
B-rep（边界表示），见Boundary representation
Back distance (*B*)（后距离），204
Back-face culling（背面消除），448~449, 463, 468, 522
Background, color attribute（背景，颜色属性），32~33
Bandwidth（带宽），137, 152
Baseline , text（基线, 文本），34
Basis matrix（基矩阵），331~332, 337, 344, 367
Batch, screen updates（屏幕批量更新），281
Bernstein polynomials（Bernstein多项式），337~338
Bézier curves（Bézier曲线），见Splines, Bézier curves
Bicubic surface（双三次曲面），323, 351
Bicubic surfaces, drawing（画双三次曲面），355~357
Bidirectional reflectivity（双向反射率）
  diffuse（双向漫反射率），490
  specular（双向镜面反射率），490
Bilevel CRT（二值阴极射线管），见Bilevel display

Bilevel display（二值显示器）10, 400~401, 403
Binary space-partitioning (BSP)tree（二元空间划分树）
regularized Boolean set operations（正则的布尔集合运算）, 380
shadow algorithm（阴影算法）, 505
for solid modeling（实体造型）, 386~388
visible-surface determination（可见面的判定）, 467~468
Binocular disparity（双眼视差）, 437~438
BitBlt（位图数据块传送）。另见CopyPixel
implementation（实现）, 119
Bitmap（位图）, 7, 65。另见Pixmap
characters（字符）, 116
graphics(图形), 7
offscreen（画外, 屏幕外）, 见Canvas, offscreen
pattern（图案）, 30~32
Blanking（留空）, 9
Blending functions, curves（调配函数）, 332~334, 338, 343~347
Boldface（粗体, 黑体）, 见Character
Boundary representation（边界表示）, 377~378, 380, 389~391
regularized Boolean set operations（正则布尔集合运算）, 371, 374, 376, 389
Bounding box（边界盒）, 447, 463。另见Extent
Box filter（盒式过滤器）, 见Filter, box
Bresenham, circle scan conversion（Bresenham, 圆扫描转换）, 见Scan conversion, midpoint circle
Bresenham, line scan conversion（Bresenham, 线扫描转换）, 见Scan conversion, midpoint line
Brightness, of light（光的辉度）, 402
Brush（画刷）
orientation（方向）, 97
shape（形状）, 97
Bsp tree（BSP树）, 见Binary Space-Partitioning tree
B-spline curves（B样条曲线）, 见Splines, B-spline curves
B-spline surfaces（B样条曲面）, 见Splines, B-spline surfaces
Button mask, locator（定位器按钮标志）43~44

C

CAD, 见Computer-aided design
Calligraphic display（书法显示器）, 8
CAM, 见Computer-aided manufacturing
Camera（照相机, 摄像机）
synthetic(合成), 193~194, 253~254
viewing（观察）, 235
Canvas（画布）, 49
context(环境, 上下文), 50
offscreen（屏外）, 68
screen(屏幕), 50
state(状态), 50
Cartography（制图学）, 3
Cathode-ray tube (CRT)（阴极射线管）, 397, 399。另见Flat tension mask, Flicker, Focus
delta-delta, 137
monochromatic(单色), 135
precision-in-line delta(在线精度delta), 137
shadow-mask（阴罩）, 137
Cell decomposition（单元分解）, 381
Center of projection (COP)（投影中心）, 195, 203
Center of window (CW)（窗口中心）, 203
Central Structure Storage（中央结构存储器）, 247
Character（字符）
alignment（排列）, 117
baseline(基线), 34
boldface（黑体）, 118
descender（下降部分, 字母下部）, 34, 117
font（字体）, 33, 117
italic（斜体）, 33, 118
recognition（识别）, 311
roman（罗马）, 118
typeface（字型）, 116
width（宽度）, 33, 117
Charge-coupled device (CCD)（电荷耦合器件）, 157
Choice logical device（选择逻辑设备）, 37, 153, 301
Choice set（选择集）, 305, 308~311
Chord, locator button state（和弦, 定位器按钮状态）, 40
Chroma, color（色度）, 402
Chromaticity, color（色度）, 402, 407~410, 415
Chromaticity coordinates（色度坐标）, 406~410
Chromaticity diagram（色度图）, 见CIE chromaticity diagram
CIE（Commission Internationde de l'Éclairage）（国际照明委员会）, 407~410, 415
CIE chromaticity diagram（CIE色度图）, 407~410, 415
CIE color model（CIE颜色模型）, 419
CIE primaries（CIE基色）, 408
Circles（圆）, 见Scan conversiom, circles
Click and drag interaction（点击与拖动交互操作）, 316
Clip rectangle（裁剪矩形）, 52
Clipping（裁剪）, 69
analytical（解析）, 100
characters（字符）, 116
Cohen-Sutherland line algorithm（Cohen-Sutherland直线算法）, 103~107
endpoints（端点）, 102
in homogeneous coordinates（齐次坐标中的）, 229
Liang-Barsky line algorithm（梁友栋-Barsky直线算法）, 111, 125
lines（线）, 101~111
Nicholl-Lee-Nicholl, line algorithm（Nicholl-Lee-Nicholl线算法）, 111, 125
polygon（多边形）, 112~113

Sutherland-Hodgman polygon-clipping algorithm（Sutherland-Hodgman多边形裁剪算法）, 112~115
text string（文本串）, 117
3D（三维）, 195, 235, 445~446, 448, 451, 475
3D Cohen-Sutherland line algorithm（三维Cohen-Sutherland线算法）, 227~229
3D Cyrus-Beck（三维Cyrus-Beck算法）, 227
2D primitives in a raster world（光栅世界的二维图元）, 100~115
2D raster graphics（二维光栅图形学）, 52
Clipping plane（裁剪平面）
back（yon）（后）, 205, 212~213, 222, 227
front（hither）（前）, 204~205, 222
Clustered-dot ordered dither（聚点有序抖动）, 399
CMY color model（CMY颜色模型）, 411
CMYK color model（CMYK颜色模型）, 412
Coding, visual（编码, 可视）, 317
Cohen-Sutherland line-clipping algorithm（Cohen-Sutherland线裁剪算法）, 见Clipping, Cohen-Sutherland line algorithm
Coherence（相关性）
area coherence（区域相关性）, 443
depth coherence（深度相关性）, 444
edge coherence（边相关性）, 85, 444
face coherence（面相关性）, 444
frame coherence（帧相关性）, 444
implied edge coherence（隐含边相关性）, 444
object coherence（对象相关性）, 443
scan-line coherence（扫描线相关性）, 85, 444
span coherence（跨度相关性）, 455
spatial（空间相关性）, 85
Color（颜色）, 见Dominant wavelength, Hue, Luminance
Color coding（颜色编码）, 419
Color deficient（色盲）420
Color gamuts（颜色域）, 409~410
Color harmony（色彩融合）, 419
Color interpolation（颜色插值）, 418
Color map（色彩图）, 见Video look-up table
Color matching functions（颜色匹配函数）, 405~406
Color models（颜色模型）, 410。另见CIE, CMY, HSV, RGB, YIQ
Color table（颜色表）, 29。另见Video look-up table
Color usage rules（颜色使用规则）, 418
Colorimetry（色度学, 比色计）, 403
Commission Internationale de l'Éclairage（国际照明委员会）, 见CIE
Commutativity, matrix operations（交换性, 矩阵运算）, 163, 177
Complementary colors（补色）, 409, 411~412
Composite interaction task（组合交互作业）, 见Interaction tasks
Composite video（复合视频）, 152
Composition（合成）, 见Transformation composition
Computer-aided design (CAD)（计算机辅助设计）, 5, 16, 247
Computer-aided manufacturing (CAM)（计算机辅助制造）, 7
Cone filter（锥形滤波器）, 124
Cone receptors in eye（锥形视觉接收器）, 404
Cones（锥体）, 488
Connectivity（连通性）, 16
Constraint, in line drawing（画线中的约束）, 316
Constructive solid geometry（CSG）（构造实体几何）, 388, 458
Continuity（连续性）, 见Geometric continuity, Parametric continuity
Contouring, intensity（轮廓线, 亮度）, 398~399
Control grid, of CRT（阴极射线管的控制栅格）, 136
Control to display ratio(C/D ratio)（控制-显示比率）, 300
Convergence of electron beam（电子束的聚焦性）, 136
Conversion between color models（颜色模式间转换）, 410~417
Convex hull（凸包）, 338~339, 343, 347
Cook-Torrance, 见Illumination, Cook-Torrance
Coordinate system（坐标系）
application（应用）, 58~59, 234
camera（照相机）, 235
device（设备）, 235
eye（眼睛）, 194, 235
left-handed（左手）, 235
local（局部）, 234
logical device（逻辑设备）, 235
modeling（模型）, 234
normalized device（规格化设备）, 235
normalized projection（规格化投影）, 235
object（物体）, 234
problem（问题）, 234
raster（光栅）, 235
right-handed（右手）, 180, 202
screen（屏幕）, 235, 313
in SRGP（在SRGP中）, 52~53
(u, v, n), 202
(u, v, VPN), 202, 235
viewing-reference(VRC)（观察参考）, 202
view-up vector（VUP）(上方向量), 202
world（世界）, 177, 234
Coordinate-system representation（坐标系表示）, 71
CopyPixel, 96
Core Graphics System（核心图形系统）, 12, 235
Correlation.（相关）, 见Pick correlation
CRT（阴极射线管）, 见Cathode-ray tube
CSG, 见Constructive solid geometry

Cuberille，382
Cubic curve（三次曲线），322
Current position (CP)（当前位置），146~147
Cursor, 3D（游标），454
Curved surfaces（曲面）。另见Surface patch
display methods（显示方法），355~356
tesselation（网格化），522
Curves（曲线）。另见Splines
parametric polynomial（参数多项式），322
parametric cubic（三次参数），328 ~ 350
Curve fitting（曲线拟合），348~349。另见Splines, Bézier curves, fitting data with
Cyrus-Beck line-clipping algorithm（Cyrus-Beck线裁剪算法），107

D

D(microfacet distribution function)（微面元分布函数），491
d_L(distance from point source to surface)（点源到曲面的距离），482
Damage repair（破损修复），33
Data tablet（数据输入板），见Tablet
DataGlove（数据手套），302
DDA, 见Digital differential analyzer
Decision variable（判定变量）
for circle scan conversion（圆扫描转换），82
for line scan conversion（线扫描转换），74
Deflection coils（偏转线圈），135
Degrees, rectangular（角度，矩形），27
Depth clipping（深度裁剪），429
Depth cueing（深度提示），428, 439, 483
Depth of field（域的深度），433
Depth-sort algorithm（深度排序算法），466, 473~474
Descender, character, 见Character, descender
Desktop（桌面）
metaphor（隐喻），297
windows(窗口)，7
Determinant（行列式），见Matrix, determinant of
Device-independence（设备无关性）
graphics（图形），12
interface（接口），66
Dialogue box（对话框），315
Diffuse reflection（漫反射），479
coefficient（k_d）（系数），480
Digital differential analyzer（数字微分分析器），72
Digitize（数字化），305, 308
Dimension, fractal（维数，分形），359
Direct manipulation（直接操纵），8
pointing and clicking（指点并单击），8
user interfaces（用户界面），298
Direction of projection (DOP)（投影方向），196, 198, 203
Direction of reflection ($\bar{R}$)（反射方向），485
Dispersion（散射，离差），507
Display（显示）
controller（控制器），8, 67~68
list（列表），149
primitives（图元），8
Display coprocessor（显示协处理器），见Graphics display processors
Display devices, raster（显示设备，光栅），21
Display list storage（显示列表存储），148
Display processing unit（显示处理单元（DPU）），见Display controller
Display program（显示程序），见Display list
Display traversal（显示遍历），253
attribute inheritance（属性继承），273, 275
implementation（实现），290
modeling transformation（模型变换），269
optimization（优化），291
viewing transformation（观察变换），253~256
Do what I mean (DWIM), 306
Dominant wavelength, color（主波长，颜色），403~405, 407~409
Dot product（点积），164
Dot size（点尺寸），130
Double-buffering（双缓冲），150, 291
Dynamic range, intensity（亮度变化范围），397, 399
Dynamics（动力学），434
geometric modeling（几何造型），265
motion（运动），14, 277
update（更新），14, 277

E

E_i (incident irradiance)（入射光的辐照度），490
Echo（回应），见Feedback
Edge coherence（边相关性），85, 90
Edge table（边表），454
Editing, of structure network（结构网络的编辑），240, 248, 277~281
Electroluminescent (EL) display（电致发光显示），140
Electromagnetic energy（电磁能），404
Electron gun（电子枪），140, 145
Element, structure network（元素，结构网络），247, 249, 277
Ellipse（椭圆）
arc（弧），26~27
Energy distribution（能量分布），见Spectral energy distribution
Ergonomics（人类工程学），见Human factors
Error diffusion, Floyd-Steinberg（误差扩散），401
$\eta_{i\lambda}$, $\eta_{t\lambda}$ (indices of refraction)（折射率），505
Euler operators（欧拉算子），379~380
Euler's formula（欧拉公式），378~379

Event（事件）, 17
SRGP input mode（SRGP输入模式）, 37
Event queue（事件队列）, 38~39
Event-driven（事件驱动）
interaction（交互）, 38~39
loop（循环）, 17
Excitation purity, color（色纯度, 颜色）, 403
Explicit functions（显函数）, 328
Extent（范围）, 446
minmax（最小最大）, 447~448
text（文本）, 33
3D object（三维物体）, 291, 295
Eye（视线）, 400, 448, 459
Eye-hand coordination（眼手协调）, 299

F

F_λ (Fresnel term)（菲涅尔项）, 491
f_{att} (light-source attenuation factor)（光源衰减因子）, 482
Feedback, input devices（输入设备反馈）, 47
Filling（填充）
algorithms（算法）, 126
pattern（图案）, 94
polygon（多边形）, 87
rectangle（矩形）, 85
Film recorder（胶片记录器）, 129, 134
Filter（滤波器）
box（盒式）, 123
cone（锥式）, 124
function（函数）, 123
highlighting/visibility（高亮度/可见性）, 289
support（支集）, 123
Fitts' law（Fitts法则）, 310
Flaps（挡板）, 488
Flat tension mask CRT（平面拉伸荫罩CRT）, 404~405
Flicker, Focus, CRT（闪烁, 焦点, CRT）, 8, 136, 410~411
Flight simulator（飞行模拟器）, 14
Floodlight（泛光灯）, 488
Fluorescence, phosphor（荧光, 荧光物质）, 136
Flux（通量）, 489。另见Irradiance
Focus, CRT（CRT焦点）, 135~136
Font（字体）, 见Character
Font cache（字体高速缓存）, 116
Foot switch（脚踏开关）, 301
Footprint（足迹）, 99
interaction device（交互设备）, 299
Foreshortened surface area（透视缩小曲面面积）, 489
Fractal dimension（分形维数）, 见Dimension, fractal
Fractal models（分形模型）, 358~363
Frame buffer（帧缓存）, 10, 142~145
Fresnel term（菲涅尔项）, 见F_λ
Front distance（F）（前距离）, 204~205, 222
Function key（功能键）, 157, 301, 311
Fusion frequency, critical（临界停闪频率）, 136

G

G(geometrical attenuation factor)（几何衰减因子）, 491
Gamma correction（Gamma校正）, 397
Generalized cylinder（广义柱面）, 见Sweep
Genus of a polyhedron（多面体的亏格）, 379
Geometric continuity（几何连续性）, 330~332
Geometric extents（几何范围）, 见Extent
Geometric modeling（几何造型）, 见Structure hierarchy
interactive（交互式）, 268
object hierarchy（对象层次）, 243
Geometry matrix（几何矩阵）, 331~333, 336~337, 343, 351~353
GKS-3D, 12
Global transformation matrix（全局变换矩阵）, 270
Gouraud, 见Shading, Gouraud
Graftals, 363
Grammar-based models（基于文法的模型）, 363~366
Graphical languages（图形语言）, 见Animation, graphical languages
Graphics（图形）
display processors（显示处理器）, 141
subroutine library（子程序库）, 17
subroutine package（子程序包）, 17, 239
Graphics system（图形系统）
input transformation（输入变换）, 15
output transformation（输出变换）, 15
Graphics workstation（图形工作站）, 见Workstation
Group technology（成组技术）, 375

H

$\bar{H}$(halfway vector)（中间向量）, 487
Halftone pattern（半色调图案）, 52
Halftoning（半色调）, 399~400
Halfway vector（$\bar{H}$）（中间向量）, 487
Handles for user interaction（用户交互柄）, 317
Head-mounted display（头盔显示器）, 303
Hermite curves（Hermite曲线）, 332。另见Splines
Hermite surfaces（Hermite曲面）, 351
Hexcone HSV color model（六棱锥HSV颜色模型）, 414~415, 423
Hidden-line determination（隐藏线判定）, 见Visible-line determination
Hidden-surface elimination（隐藏面清除）, 见Visible-surface determination
Hierarchical menu selection（层次菜单选择）, 309
Hierarchical object selection（层次对象选择）, 306~307
Hierarchy（层次, 层次结构）, 450, 463。另见Display traversal, Ray tracing

data（数据）, 293
limitations in modeling（建模的限制）, 292
object（对象）, 306
object modeling（对象建模）, 243~245
Highlighting, in geometric model（几何模型的加亮显示）, 289
Hither clipping plane（裁剪平面）, 见Clipping plane, front
Homogeneous coordinates（齐次坐标）, 171
HOOPS, 13, 239, 293~294
HSB color model（HSB颜色模型）, 见HSV color model
HSV color moder（HSV颜色模型）, 413
Hue, color（色调）, 402, 410, 413, 415~416, 419
Human factors（人的因素）, 36

I

$I_{dc\lambda}$ (depth cue color)（深度提示颜色）, 484
I_i (incident radiance)（入射辐射光亮度）, 490
I_p(point light source intensity)（点光源强度）, 480
I_r (reflected radiance)（反射辐射光亮度）, 470
Illuminant C（发光物C）, 407
Illumination（光照）。另见Light Source
Cook-Torrance（Cook-Torrance模型）, 489
equation（方程）, 478
global（全局）, 478, 509
local（局部）, 509
model（模型）, 477
Phong（Phong光照模型）, 485
physically based models（基于物理的模型）, 489
Torrance-Sparrow （Torrance-Sparrow模型）, 490
Image, scaling（图像，缩放）, 60~61
Image irradiance（图像辐照度）, 490
Image-precision（图像精度），见 Visible-surface determination
Immediate-mode graphics（立即模式图形）, 247
Implicit equations（隐式方程）, 328
Inbetweening（插值）, 436
Incremental methods（增量方法）, 70
Input（输入）, 见Interaction handling
Input devices（输入设备）153, 297~298。另见 Interaction tasks, Logical input device
Input pipeline（输入管线，输入流水线）, 66
Inside（内部）, 见Odd-parity rule, Filling algorithms
Instance, object hierarchy（实例，对象层次）, 245
Instance block（模块示例）, 279~281
Intensity（亮度）
light（光）, 396, 399
of line as function of slope（直线的亮度是斜率的函数）, 79
radiant（辐照）, 489
resolution（分辨率）, 399
Interaction（交互）, 见Logical input device
Interaction handling（交互处理）, 17~18, 36, 45
sampling vs. event（采样与事件）, 38~39
in SPHIGS（SPHIGS中的交互处理）, 282~285
in SRGP（SRGP中的交互处理）, 45~47
Interaction tasks（交互任务）, 298, 304~314
composite（复合）, 314~318
position（定位）, 298, 304
quantify（量化）, 298, 311
select（选择）, 298, 305, 308
text（文本输入）, 298, 311
3D（三维）, 312~314
3D rotation（三维旋转）, 313
Interaction techniques（交互技术）, 298, 304~305, 318
color specification（颜色规范）, 417
Interaction toolkits（交互工具箱）, 318~319
Interdot distance（点间距离）, 130
Interlaced scan（隔行扫描）, 151
International Standards Organization（国际标准化组织）, 见ISO
Interpolation（插值）。另见Shading
color（颜色）, 418
Intersection, external（交集，外部）, 107
InterViews, 319
Inverse（逆）, 见Matrix, inverse of
Irradiance（辐照度）, 490
ISO (International Standards Organization)（国际标准化组织）, 12, 283
Italic（斜体）, 见Character, italic
Item buffer（分项缓存）, 518

J

Jaggies（锯齿状图形）。另见Aliasing, Antialiasing
Joystick（游戏杆）, 155, 298~301
JPL（Jet Propulsion Lab）（美国CIT的喷气推进实验室）, 425
Julia-Fatou set,（Julia-Fatou集合）, 359
Just-noticeable color difference（可察觉的颜色差）, 405

K

k_a(ambient-reflection coefficient)（环境反射系数）, 479
k_d(diffuse-reflection coefficient)（漫反射系数）, 480
k_i(intrinsic color)（固有颜色值）, 478~479
k_s(specular-reflection coefficient)（镜面反射系数）, 485
k_t(transmission coefficient)（透射系数）, 505
Kerning(字距调整), 117
Keyboard（键盘）, 298, 301
alphanumeric（字母数字）, 156
coded（编码）, 156
logical input device（逻辑输入设备）, 37, 42, 153
unencoded（非编码）, 156

Key-frame（关键帧），436
Knot, cubic curve（三次曲线结点），342

L

$\bar{L}$(vector to light L)（到光源L的向量），478
Label, structure element（标签，结构元素），278
Lambertian reflection（朗伯反射），479
Lambert's law（朗伯定律），480
Lateral inhibition（横向抑制），493
LCD（液晶显示器），见Liquid-crystal display
Length, of a vector（向量长度），164
L-grammars（L文法），363
Liang-Barsky line-clipping algorithm（梁友栋-Barsky裁剪算法），见Clipping, Liang-Barsky line algorithm
Light（光），见 Ambient light, Diffuse reflection, Illumination
Light-emitting diode (LED)（发光二极管），302~303
Light source（光源）。另见Illumination
  attenuation（衰减），482
  attenuation factor（f_{att}）（衰减系数），482
  colored（有色光源），482
  cones（光锥，锥体），488
  directional（定向光），431, 481
  distributed（分布），431
  extended（扩展），431
  flaps（挡板），488
  floodlight（泛光灯），488
  overflow（溢出），489
  point（点光源），431, 479
  spotlight（聚光灯），488
  Warn controls（警告控制），487
Light valve projection system（光阀投影系统），140
Lighting（发光光照）。另见Illumination
Lightness, color（明度），402
Line（线，直线），另见Scan conversion
  style（型），27~29
Line drawing（线框图），8
Linear combination（线性组合），163
Linguistic interaction task（语言交互任务），304
Liquid-crystal display (LCD)（液晶显示器），129, 138
List-priority algorithms（列表优先级算法），465, 468。另见Depth-sort algorithm, Binary space-partitioning tree
Local control, cubic curves（三次曲线的局部控制），342
Local transformation matrix（局部变换矩阵），258
Locator（定位器），17
Locator logical device（定位逻辑设备），153
  absolute（绝对），299
  continuous（连续），299
  direct（直接），299
  discrete（离散），299
  indirect（间接），299
  relative（相对），299
  3D（三维），282, 301~304
  2D（二维），37~47
Logical input device（逻辑输入设备），153, 298
Look-up table (LUT)（查找表）。另见Video look-up table
Luminance, color（光强度），396, 403
Luminous efficiency function（光效率函数），404
Luminous energy（发光能量），407
Luxo, Jr., 433

M

Mach bands（马赫带效应），439
Macintosh, 310~311, 319
Mandelbrot set（Mandelbrot 集），358~361
Marker, output primitive（标记，输出图元），24
Masks（屏蔽），52
Master, object hierarchy（宿主，对象层次结构），245
Material properties（材质属性），431
Matrix（矩阵）
  determinant of（的行列式），166
  identity（单位），166
  inverse of（逆），167
  multiplication of（乘法），165~166, 167
  transpose of（转置），166~167
Matrix addressing of display（显示器的矩阵寻址），139
Measure, logical input devices（度量，逻辑输入设备），38, 40~44
Menu（菜单）
  bar（条），46
  headers（头），46
  body（体），46
Menus（菜单）
  appearing（呈现），309
  hierarchical（分级），309
  pop-up（弹出），309~310
  pull-down（下拉），309~310
  static（静态），309
  tear-off（浮动），309
Metafile（元文件），289~290
Microfacets（微面元），490
Microsoft（微软公司），310, 311
Modality（模态），27
Mode, input devices（模式，输入设备），40
Modeling（建模，造型），240
Modeling（建模，造型），见Geometric modeling
Modeling transformation（模型变换），见Transformation, modeling
Molecular modeling（分子建模），425
Monochrome（单色），见Bilevel display

Motion blur（运动模糊），433
Mouse（鼠标），8, 154, 298~300, 304, 309~310, 310, 313, 315。另见Locator
mechanical（机械），154
optical（光学），154
Mouseahead（鼠标在前），40
Multimedia systems（多媒体系统），5
Multiple control points, curves（曲线多控制点），344~345, 348
Munsell color-order system（Munsell颜色排序系统），402

N

$\bar{N}$(surface normal)（曲面法线），479
n (specular reflection exponent)（镜面反射指数），485
Name set, PHIGS（PHIGS的名称集），289
NASA（美国宇航局），425
National Television System Committee(NTSC)（美国国家电视系统委员会），151~152, 413
Natural cubic spline（自然三次样条），342
Natural phenomena（自然现象），358
Necker cube illusion（Necker立方体幻觉），426
Newell-Newell-Sancha algorithm（Newell-Newell-Sancha算法），见Depth-sort algorithm
NeXT, 310, 312
Nicholl-Lee-Nicholl algorithm（Nicholl-Lee-Nicholl算法），见Clipping, Nicholl-Lee-Nicholl line algorithm
Nonspectral color（非光谱颜色），409
Nonuniform, nonrational B-splines（非均匀非有理B样条），345~348
Normal（法线）
to bicubic surface（双三次曲面），354
to plane（平面），183, 326
to quadric surface（二次曲面），357
to surface（曲面），521
Normalized projection coordinates(NPC)（规格化投影坐标），205
Normalizing transformation（规格化变换）
parallel（平行），217~222
perspective（透视），222~224
NPC，见Normalized projection coordinates
NTSC（国家电视系统委员会），见National Television System Committee
NURBS（非均匀有理B样条）见Splines， Nonuniform, rational B-splines (NURBS)

O

$O_{d\lambda}$(object diffuse color)（物体漫反射颜色），482
$O_{s\lambda}$(object specular color)（物体镜面反射颜色），486
Object buffer（物体缓存），454
Object hypothesis（目标假设），426
Object modeling （物体建模），见Geometric modeling
Object-precision（对象精度），见Visible-surface determination
Octree（八叉树）
neighbor finding（查找邻节点），386
regularized Boolean set operations（正则布尔集合运算），385
rotation（旋转），386
Odd-parity rule（奇数规则），30
Opacity（不透明性），505
OPEN LOOK（ Sun Microsystem公司提供的一种图形用户界面），319
Open Software Foundation (OSF)（开放软件基金会），319
OpenGL, 13, 239, 293~294
Ordered dither（有序抖动），399
Ostwald color-order system（Ostwald颜色排序系统），403
Outcode, clipping（外码），104~107
Output pipeline（输出流水线），69。另见Rendering
Output primitives（输出图元）
geometric modeling（几何建模），250
raster graphics （光栅图形），22~26
respecification（重新定义），59

P

Painter's algorithm（画家算法），466
Painting, implementation（实现着色），41
PAL television standard（PAL制式电视标准），152
Palette（调色板），36
PANTONE MATCHING SYSTEM， 402
Parallel projection（平行投影）
front（前向），198, 201, 211
oblique（斜），198, 200, 201
orthographic（正交），198, 201
plan（计划）， 198
side（侧面），198, 201, 212
top（顶部），198, 201, 212
Parallelogram rule（平行四边形法则），163
Parametric bivariate polynomial surface patches（参数双变量多项式曲面片），322。另见Splines
Parametric continuity （参数连续性），330~331
Parametric cubic curves（三次参数曲线），见Curves, parametric cubic
Parametric polynomial curves（参数多项式曲线），见Curves, parametric polynomial
Parametric representation, in line clipping（参数表示），107
Parametric surfaces（参数曲面），322, 351
Parametric velocity（参数速率），331
Pattern, output attribute（图案, 输出属性），31~33
Pattern filling（图案填充），见 Filling, pattern
Pattern mapping（模式映射），见 Surface detail

Pattern recognition interaction technique（模式识别交互技术），310~311
Pels（图像单元），7
Pen style（笔型，画笔类型），99
Perceived intensity of light（光敏强度），396
Persistence, phosphor（余辉，荧光物质），136
Perspective foreshortening（透视缩短，透视缩小），196
Perspective projection（透视投影）
one-point（一点），197, 207~208, 215
three-point（三点），198, 215~216
two-point（两点），198, 208~209, 215
Perspective transformation（透视变换），见Transformation, perspective
Phantom vertices, cubic curves（三次曲线假设顶点），345
PHIGS，12。另见SPHIGS
PHIGS Plus, 13
Phong（Phong光照模型），见Illumination, Phong, Shading, Phong
Phosphor（磷光物质，荧光物质），135~136, 398, 405, 410
Phosphorescence（磷光），136
Photometer（光度计），397
Photorealism（照片真实性），423~425
rendering（绘制），19
Pick（拾取）
correlation（关联拾取），45~47
correlation implementation（关联拾取实现），284
correlation object hierarchy（关联拾取对象层次），282
identifier（标识符），284
logical device（逻辑设备），153
logical input device（逻辑输入设备），37
point（点），103
window（窗口），103
Picture hierarchy（画面层次），见Hierarchy, object
Piecewise continuous polynomial（分段连续多项式），322, 328, 332
Pitch, shadow mask CRT（阴罩CRT的间距），138
Pitteway, 73
Pixar, 235, 433
PixBlt（像素块传送指令），142, 152。另见BitBlt
Pixel（像素），7
geometry（几何），122
replication（复制），60, 99
Pixmap（像素图），10, 65, 141, 146, 152。另见Canvas, offscreen
pattern(图案), 32
Planar geometric projection（平面几何投影），196
Plane（平面）
equation（平面方程），166, 183, 325~327
Plasma panel display（等离子平板显示器），140
Plotter（绘图仪）
drum（鼓式），132
desktop（桌面），132
flatbed（平板），132
ink-jet（喷墨），132
Point light sonrce（点光源），见Light source, point
Point of evaluation（估值点），82
Polhemus, 3D digitizer（Polhemus三维数字化仪），302~303
Polling（轮询），见Sampling
Polygon（多边形）
interior test（内部测试），29
Polygon clipping（多边形裁剪），见Clipping, polygon
Polygon mesh（多边形网格）
consistency（一致性），322~325
Polygon table（多边形表），455
Polyhedron（多面体）
simple（简单多面体），378
SPHIGS, 251
Polyline（折线），23
Portability（可移植性），12, 239
application programs（应用程序），254
Positioning interaction task（定位交互任务），见Interaction tasks
PostScript（一种页面描述语言），59, 133, 349
Potentiometer（电位器），301, 311
Primaries, color（基色），410, 411~412
Primitive instancing（基本实体举例法），375
Primitives（图元），见Output primitives
Printer（打印机）
dot-matrix（点阵），131
int-jet（喷墨），133
laser（激光打印机），132
thermal-transfer（热传导），133
sublimation dye transfer(热升华印染传导打印机), 133
Priority, display（优先级，显示），256
Processing mode, keyboard（处理模式，键盘），42
Progressive refinement（逐步细化），524。另见Radiosity
Projection（投影），194
axonometric parallel（轴测平行），199
isometric parallel（等轴平行），200
oblique parallel（斜平行），200
orthographic parallel（正平行），194, 198
parallel（平行），196
perspective（透视），196
Projection implementation（投影实现）
parallel（平行），214~222
perspective（透视），222~227
Projection matrix（投影矩阵）
orthographic（正交），215
perspective（透视），214~215
Projection plane（投影平面），195
Projection reference point (PRP)（投影参考点），203
Projector（投影线），195
Pruning（修剪），292

Pseudorealism, rendering（伪真实感，绘制），见Photorealism
Pulldown menu（下拉菜单），见Menus, pulldown
Purity（纯度），见Excitation purity

Q

Quadric surface（二次曲面），323, 357
Quadrilateral mesh（四边形网格），见Polygon mesh
Quadtree（四叉树），450, 471
  neighbor finding（邻节点查找），386
Quantity interaction task（定量交互任务），见Interaction tasks
QuickDraw, 13, 98

R

$\bar{R}$(direction of reflection)（反射方向），485
Radiance（辐射光亮度），489
Radio button interaction technique（单选按钮交互技术），310
Radiosity (B)（辐射度），515
Radiosity methods（辐射度方法）
  ambient term（环境项），521
  z-buffer（z缓存），521
  color bleeding（渗色），516
  delta form factors（Δ形状因子），518
  form factor ($F_{i\text{-}j}$)（形状因子），515
  form factor reciprocity relationship（形状因子相互性关系），515
  gathering（聚集），520
  Gauss-Seidel iteration（Gauss-Seidel迭代），516
  hemicube（半立方体），518
  progressive refinement（逐步细化），519
  radiosity equation（辐射度方程），515
  shooting（发射），520
  vertex radiosities（顶点辐射度），516
Random scan（随机扫描），8
Raster（光栅），8
  display（显示器），145~150。另见Cathode-ray tube
Raster graphics package（光栅图形软件包），13, 21~22, 49, 60
Raster image processor（光栅图像处理器），69
Raster lines（光栅行），8
Raster operation（光栅操作），149。另见BitBlt, PixBlt, Write mode
Rasterization（光栅化），424。另见Scan conversion, Shading
RasterOp, 54~56, 147, 309。另见Raster operation
Raster-scan generator（光栅扫描生成器），143
Ray, eye（光线，视线），459
Ray casting（光线投射），380, 459。另见Ray tracing in b-rep Boolean set operations(在b-rep布尔集合运算中), 380
Ray tracing（光线跟踪），459~465, 510~514
  bounding volume（包围体），463
  computing intersections（求交），460~462, 463
  efficiency（效率），462~465
  hierarchy（层次结构），463
  mass properties（质量属性），476
  numerical precision problems（数值精度问题），514
  octrees（八叉树），464
  polygons（多边形），461~462
  primary rays（主光线），511
  ray tree（光线树），511
  recursive（递归），512
  reflection rays（反射线），511
  refraction rays（折射线），511
  secondary rays（从属光线），511
  shadow rays（阴影光线），510
  shadows（阴影），510
  spatial partitioning（空间划分），463~464
  spheres（球体），460~461
  surface normal（曲面法向量），461
Realism（真实感），见Photorealism
Rectangle write（矩形写），95
Reflected light（反射光），402
Reflection（反射），432。另见Diffuse reflection, Direction of reflection, Specular reflection
Refraction（折射），432
Refraction vector（折射向量）($\bar{T}$), 507
Refresh（刷新），8, 135~136。另见Display traversal
Refresh rate（刷新率），135, 145
Region checks, clipping（区域检测，裁剪），103
Regularized Boolean set operations（正则布尔集合运算），372~374, 376, 389。另见Constructive solid geometry
  for binary space partitioning trees（二元空间划分树），387
  for boundary representations（边界表示），380
  compared with ordinary Boolean set operations（与一般布尔集合的比较），372~373
  for octrees（八叉树），385
  for sweeps（扫掠），377
  ray tracing（光线跟踪），458
Rendering（绘制），424
Rendering, SPHIGS。另见Display traversal
  types（类型），276~277
Rendering equation（绘制方程），510
Rendering pipeline（绘制流水线，绘制流程），521
  z-buffer（z缓存），521
  global illumination（全局光照），523
  Gouraud shading（Gouraud明暗处理），521
  list-priority（列表优先级），523
  local illumination（局部光照），521
  Phong shading（Phong阴影处理），523

ray tracing（光线跟踪），524
RenderMan, 235
Replace write mode（替换写模式），95, 119
Resolution（分辨率），131, 134, 137, 138
Retained-mode graphics（保留模式图形），见SPHIGS,
RGB color model（RGB颜色模型），410
ρ(bidirectional reflectivity（双向反射率），490
ρ_d(diffuse bidirectional reflectivity)（双向漫反射率），490
ρ_s(specular bidirectional reflectivity)（双向镜向反射率），490
Right justification（正确判断），112
RIP, 见Raster image processor
Root, structure network（根，结构网络），253
Rotation（旋转）
3D（三维），181
2D（二维），175
Rotation interaction task（旋转交互任务），见Interaction tasks
Rubberband（橡皮筋线）
circle drawing（画圆），315
feedback（反馈），47~48
line drawing（画线），316
rectangle drawing（画矩形），315

S

S_i (shadow coefficient)（阴影系数），502
Sample（样本）, 17
Sample, SRGP input mode（样本，SRGP图形包输入模式），40~41
Sample-driven interaction（样本驱动交互），38~40
Sampling（采样），120~125
unweighted area sampling（未加权区域采样），120
weighted area sampling（加权区域采样），122
Saturation, color（饱和度，颜色），402~403, 407, 410, 415, 418
Scaling（缩放）。另见Image scaling
differential（不同的），169
3D（三维），181
2D（二维），169, 172~173
Scan conversion（扫描转换）。另见Rasterization
antialiased lines（反走样直线），119~125
characters（字符），116~118
circles（圆），80~85
incremental line（增量线），71~73
line（线），8, 70~80
midpoint circle（中点圆），81~85
midpoint line（中点线），73~80
outline primitives（轮廓线图元），79~80
rectangle（矩形 ），85
text strings（文本字符串），117~119
thick primitives（宽图元），97
trapezoid（梯形），93
triangle（三角形），93
triangle mesh（三角形网格），93
Scan-line algorithm（扫描线算法），91, 454, 456, 458, 473~474, 502
regularized Boolean set operations（正则布尔集合运算），458
Scanner（扫描仪），157~158
Schröder stairway illusion（楼梯幻觉），426
Scientific visualization（科学计算可视化），6, 14, 435
Scissoring（截剪），69, 100
Screen angle, printing（屏幕角度），399
Scroll（滚动），149, 308
SECAM television standard（SECAM制式电视标准），152
Second-order differences, circle scan conversion（圆扫描转换二阶差分），84
Segment storage , local（段存储，本地），149
Segments, in GKS（图形核心系统中的段），12
Select interaction task（选择交互任务），见Interaction tasks
Selection（选择）
by naming（通过命名选择），305
by pointing（点取选择），306
Self-luminous object（自发光物体），408
Self-occlusion（自遮挡），480
Self-similarity （自相似性），358
Set（集，集合）
boundary points（边界点），372
closed（闭），372
closure（闭包），371
interior（内部），372
open（开），372
regular（正则），372
regularization（正则化），372
Shade, color（色深，颜色），402
Shading（明暗处理），430。另见Rasterization
constant（flat, faceted）（恒定），492
Gouraud(intensity interpolation)（Gouraud亮度插值），431, 494
interpolated（插值的），431, 492
model（明暗模型），477
Phong(normal interpolation)（Phong模型（法向插值）），495
polygon mesh（多边形网），493
problems with interpolated shading（插值明暗处理问题），496~498
Shadows（阴影），432, 501
fake（虚假），502
region of influence（影响区域），504
scan-line（扫描线），502
shadow polygon（阴影多边形），503
shadow volume（阴影体），503

shadow volume binary-space partitioning tree（阴影体的二元空间划分树）, 505
sphere of influence（影响球）, 503
Shear（错切）
3D（三维）, 182
2D（二维）, 174
SIGGRAPH（Special Interest Group on Graphics的缩写）, 12
Simulation（模拟, 仿真）, 6
Sketchpad, 7
Sliver polygons（狭长多边形）, 90
Slow-in/slow-out（渐入/渐出）, 437
Snell's law（ Snell定律）, 509
Solid angle（立体角）, 489
Solid modeling（实体造型）, 369~393。另见Geometric modeling
features（特征）, 393
point classification（点分类）, 387
robustness（健壮性）, 392
toleranced objects（带有容差的物体）, 393
user interface（用户界面）, 392
Solid modeling representation（实体造型表示）
comparison（比较）, 390~391
conversion（转换）, 391
evaluated model（已求值模型）, 391
unevaluated model（未求值模型）, 391
Span calculation（生成计算）, 93
Spatial integration（空间积分）, 399
Spatial occupancy enumeration（空间位置枚举）, 382, 390
Spatial partitioning（空间划分）, 449~450, 463~465
adaptive（适合的）, 450
Spatial partitioning representations（空间划分表示）, 381~388
Spatial resolution（空间分辨率）, 399
Spatial subdivision（空间子分）, 见Spatial partitioning
Spatial task（空间任务）, 311
Special orthogonal matrix（特殊正交矩阵）, 173, 182
Spectral energy distribution（光谱能量分布）, 403~404
Spectral-response functions（光谱响应函数）, 404
Spectrum（谱, 频谱）, 404
Specular reflection（镜面反射）, 484~485, 506
coefficient of(k_s)（镜面反射系数）, 485
exponent(*n*)（指数）, 485
Speech recognition（语音识别）, 见Voice recognition
SPHIGS, 13, 239~295。另见Structure
interaction handling（交互处理）, 282~285
object modeling（物体建模）, 257~265
output attributes（输出属性）, 252
output primitives（输出图元）, 250~252
retained-mode graphics(保留模式图形), 247
SPHIGS, screen updating（SPHIGS, 屏幕更新）, 见 viewing operations,
Splines（样条）
Bézier curves（Bézier曲线）, 331
Bézier curves, fitting data with（用Bézier曲线拟合数据）, 347~348
Bézier surfaces（Bézier曲面）, 351
B-spline curves（B样条曲线）, 342
B-spline surfaces（B样条曲面）, 354
nonuniform, nonrational B-spline curves（非均匀非有理B样条曲线）, 345
nonuniform rational B-splines（NURBS）（非均匀有理B样条曲线）, 348
nonuniform, rational cubic polynomial curve segments（非均匀有理三次多项式曲线段）, 348
uniform, nonrational B-spline curves（均匀非有理B样条曲线）, 342
used for characters（用于字符的）, 118
Spot size（点尺寸）, 见Dot size
Spotlight（聚光灯）, 488
Sprite（精灵）, 152
Squash and stretch（挤压与拉伸）, 437
SRGP, 13
framebuffer control（帧缓存控制）, 49~58
interaction handling（交互处理）, 36~39
output attributes（输出属性）, 27~33
output primitives（输出图元）, 22~27
SRGPcopyPixel, 68, 116, 119
implementation（实现）, 119
Staging（分解）, 见Animation, staging of
Staircasing（楼梯状）, 119
Standardized object（标准化物体）, 258
Standards（标准）
graphics packages（图形软件包）, 240
State-transition diagram（状态转换图）, 307
Steradian（单位立体角）, 489
Stereo（立体）
pair（对）, 312, 437
Stereopsis（立体观测, 体视学）, 437
Stimulus-response (S-R)compatibility（刺激－响应兼容性）, 313
Storyboard（故事板）, 435
Stroke（笔画）, 8
logical input device（逻辑输入设备）, 37
Structure（结构）
editing（编辑）, 277
elision（省略）, 292
example（示例）, 265~268
hierarchy（层次）, 262~268
modeling transformation（模型变换）, 257~262, 268, 291
pick correlation（关联拾取）, 282
referral（参考结构）, 292